

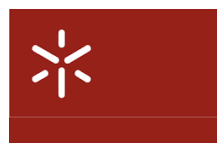


Javad Zarrin

Descoberta de Recursos para Sistemas de Escala Arbitrarias

Resource Discovery for Arbitrary Scale Systems

Programa de Doutoramento em Informática
das Universidades do **Minho, Aveiro, e Porto**



Universidade do Minho



Universidade de Aveiro



Universidade do Porto





Javad Zarrin

Descoberta de Recursos para Sistemas de Escala Arbitrarias

Resource Discovery for Arbitrary Scale Systems

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Informática, realizada sob a orientação científica dos Doutor Rui Luis Andrade Aguiar, Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e Doutor João Paulo Barraca, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro .

o júri / the jury

presidente / president

Doutor Luís António Ferreira Martins Dias Carlos

Professor Catedrático e Vice-Diretor CICECO
Departamento de Física
Universidade de Aveiro

vogais / examiners committee

Doutor Stefan Wesner

Professor and Director
KIZ - Communication and Information Centre
Universität Ulm, Germany

Doutor Rui Luis Andrade Aguiar (Orientador)

Professor Catedrático
DETI - Departamento de Electrónica Telecomunicações e Informática
Universidade de Aveiro

Doutor Orlando Manuel Oliveira Belo

Professor Associado com Agregação
Departamento de Informática, Escola de Engenharia
Universidade do Minho

Doutor Rui Manuel Rodrigues Rocha

Professor Associado
DEEC - Departamento de Engenharia Electrotécnica e de Computadores
Instituto Superior Técnico, Lisboa

Doutor José Nuno Panelas Nunes Lau

Professor Auxiliar
DETI - Departamento de Electrónica Telecomunicações e Informática
Universidade de Aveiro

agradecimentos / acknowledgments

A debt of gratitude : to my supervisor Professor Dr. Rui L.Aguiar for his precious guidance throughout this research. Without his supervision and constant help this dissertation would not have been possible. And many thanks go to my co-supervisor Professor Dr. João Paulo Barraca for his great support and valuable comments. Also... a special thank you to all the people who made the development of this work possible in their own particular way... To Ele (my father, Morad Zarrin) and Ela (my mother, Taqi Dehghani) for everlastingly helping me to be... To Armaghan for beautiful togetherness and a defamiliarized sense of being... To Fatemeh, Hamid, Gholamreza, Reza, Amir, Bahram and Saeid for boundless friendship... To Asra for beauty of hope and aspirations... To my colleagues (Professor Dr. Diogo Gomes, Professor Dr. Óscar Pereira, Dr. Daniel Corujo, André Rainho, Dr. Sérgio Figueiredo, Rui Ferreira, Carlos Guimarães, Mário Antunes, Gustavo Pires, Bruno Santos, Diogo Regateiro, José Quevedo, etc.) at ATNoG and Instituto de Telecomunicações for all their help at all times.

Palavras-chave

Descoberta de Recursos, Gestão de Recursos, Compartilhamento de Recursos, Sistemas Paralelos e Distribuídos, Sistemas de Muitos-núcleos, Orientada a Serviços, Sistemas Operacionais, Sistemas Operacionais Distribuídos, DHT, P2P, Anycasting, Computação de alto Desempenho, Grid, Cloud, Redes Sobrepostas.

Resumo

Tecnologias de Computação Distribuída em larga escala tais como Cloud, Grid, Cluster e Supercomputadores HPC estão a evoluir juntamente com a emergência revolucionária de modelos de múltiplos núcleos (por exemplo: GPU, CPUs num único die, Supercomputadores em single die, Supercomputadores em chip, etc) e avanços significativos em redes e soluções de interligação. No futuro, nós de computação com milhares de núcleos podem ser ligados entre si para formar uma única unidade de computação transparente que esconde das aplicações a complexidade e a natureza distribuída desses sistemas com múltiplos núcleos. A fim de beneficiar de forma eficiente de todos os potenciais recursos nesses ambientes de computação em grande escala com múltiplos núcleos ativos, a descoberta de recursos é um elemento crucial para explorar ao máximo a capacidade de todos os recursos heterogêneos distribuídos, através do reconhecimento preciso e localização desses recursos no sistema. A descoberta eficiente e escalável de recursos é um desafio para tais sistemas futuros, onde os recursos e as infra-estruturas de computação e comunicação subjacentes são altamente dinâmicas, hierarquizadas e heterogêneas. Nesta tese, investigamos o problema da descoberta de recursos no que diz respeito aos requisitos gerais da escalabilidade arbitrária de ambientes de computação futuros com múltiplos núcleos ativos. A principal contribuição desta tese é a proposta de uma entidade de descoberta de recursos adaptativa híbrida (Hybrid Adaptive Resource Discovery - HARD), uma abordagem de descoberta de recursos eficiente e altamente escalável, construída sobre uma sobreposição hierárquica virtual baseada na auto-organização e auto-adaptação de recursos de processamento no sistema, onde os recursos computacionais são organizados em hierarquias distribuídas de acordo com uma proposta de modelo de descrição de recursos multi-camadas hierárquicas. Operacionalmente, em cada camada, que consiste numa arquitetura ponto-a-ponto de módulos que, interagindo uns com os outros, fornecem uma visão global da disponibilidade de recursos num ambiente distribuído grande, dinâmico e heterogêneo. O modelo de descoberta de recursos proposto fornece a adaptabilidade e flexibilidade para executar consultas complexas através do apoio a um conjunto de características significativas (tais como multi-dimensional, variedade e consulta agregada) apoiadas por uma correspondência exata e parcial, tanto para o conteúdo de objetos estáticos e dinâmicos. Simulações mostram que o HARD pode ser aplicado a escalas arbitrárias de dinamismo, tanto em termos de complexidade como de escala, posicionando esta proposta como uma arquitetura adequada para sistemas futuros de múltiplos núcleos. Também contribuímos com a proposta de um regime de gestão eficiente dos recursos para sistemas futuros que podem utilizar recursos distribuídos de forma eficiente e de uma forma totalmente descentralizada. Além disso, aproveitando componentes de descoberta (RR-RPs) permite que a nossa plataforma de gestão de recursos encontre e aloque dinamicamente recursos disponíveis que garantam os parâmetros de QoS pedidos.

Keywords

Resource Discovery, Resource Management, Resource Sharing, Parallel and Distributed Systems, Many-core Systems, Service-oriented Operating Systems, Hardware Description, Distributed Operating Systems, DHT, P2P, Anycasting, Self-Configuration, High Performance Computing, Grid and Cloud Computing, Cluster Computing, Overlay Networks

Abstract

Large scale distributed computing technologies such as Cloud, Grid, Cluster and HPC supercomputers are progressing along with the revolutionary emergence of many-core designs (e.g. GPU, CPUs on single die, supercomputers on chip, etc.) and significant advances in networking and interconnect solutions. In future, computing nodes with thousands of cores may be connected together to form a single transparent computing unit which hides from applications the complexity and distributed nature of these many core systems. In order to efficiently benefit from all the potential resources in such large scale many-core-enabled computing environments, resource discovery is the vital building block to maximally exploit the capabilities of all distributed heterogeneous resources through precisely recognizing and locating those resources in the system. The efficient and scalable resource discovery is challenging for such future systems where the resources and the underlying computation and communication infrastructures are highly-dynamic, highly-hierarchical and highly-heterogeneous. In this thesis, we investigate the problem of resource discovery with respect to the general requirements of arbitrary scale future many-core-enabled computing environments. The main contribution of this thesis is to propose Hybrid Adaptive Resource Discovery (HARD), a novel efficient and highly scalable resource-discovery approach which is built upon a virtual hierarchical overlay based on self-organization and self-adaptation of processing resources in the system, where the computing resources are organized into distributed hierarchies according to a proposed hierarchical multi-layered resource description model. Operationally, at each layer, it consists of a peer-to-peer architecture of modules that, by interacting with each other, provide a global view of the resource availability in a large, dynamic and heterogeneous distributed environment. The proposed resource discovery model provides the adaptability and flexibility to perform complex querying by supporting a set of significant querying features (such as multi-dimensional, range and aggregate querying) while supporting exact and partial matching, both for static and dynamic object contents. The simulation shows that HARD can be applied to arbitrary scales of dynamicity, both in terms of complexity and of scale, positioning this proposal as a proper architecture for future many-core systems. We also contributed to propose a novel resource management scheme for future systems which efficiently can utilize distributed resources in a fully decentralized fashion. Moreover, leveraging discovery components (RR-RPs) enables our resource management platform to dynamically find and allocate available resources that guarantee the QoS parameters on demand.

Table of Contents

Table of Contents	i
List of Acronyms	v
List of Figures	ix
List of Tables	xv
List of Algorithms	xvii
1 Introduction	1
1.1 Background and Context	1
1.2 Motivation	2
1.3 Problem Statement	3
1.4 Research Aims and Objectives	4
1.5 Tools and Methodology	5
1.6 Thesis Organization	5
1.7 Novel Contributions	7
2 Discovery Protocols and Large Scale Systems	9
2.1 Introduction	9
2.2 Relevant Discovery Aspects	10
2.3 Underlying Aspects	11
2.3.1 Architectures For Resource Discovery	11
2.3.2 Computing Environments	14
2.3.3 Ontology and Resource Description	34
2.3.4 Clustering Approaches	36
2.4 Design Aspects	38
2.4.1 Bootstrapping	38
2.4.2 Discovery Strategies	39
2.4.3 Search Algorithms	39
2.4.4 Packet Propagation	48
2.4.5 Information Delivery	49
2.4.6 Query Termination	50
2.5 Evaluation Aspects	52
2.5.1 Efficiency	53
2.5.2 Scalability	53

2.5.3	Reliability	55
2.5.4	Flexibility	57
2.5.5	Heterogeneity	59
2.6	Comparison	60
2.7	Resource Management for Large Scale Systems	63
2.7.1	High Performance Computing Systems	63
2.7.2	Grid Computing Systems	64
2.7.3	Cloud Computing Systems	64
2.7.4	Manycore Systems	65
2.8	Conclusion	65
3	Hierarchical Resource Description	67
3.1	Introduction	67
3.2	Hierarchical Layers	68
3.3	Information Transmission	72
3.4	Syntaxes and Description Examples	75
3.4.1	Defining Layers and Attributes	75
3.4.2	Defining Queries	76
3.4.3	Defining Systems	80
3.5	Coding Efficiency	85
3.6	Conclusion	91
4	Hybrid Adaptive Resource Discovery	93
4.1	Introduction	93
4.2	Requirements and Design Principles	94
4.2.1	Assumptions and Definitions	94
4.2.2	Resource Description	96
4.2.3	System Architecture	96
4.3	Self-organization and Self-configuration	99
4.3.1	Description of Algorithm	99
4.3.2	Evaluation of the Group Creation Algorithm	109
4.4	Resource Discovery in Many-Core Systems	113
4.5	HARD Mechanisms	118
4.5.1	Initialization Phase	118
4.5.2	Data Structures	119
4.5.3	Resource Requester and Resource Information Provider	119
4.5.4	Dynamic Aspects	121
4.5.5	Communication Events	122
4.6	HARD Algorithms	125
4.6.1	LN Algorithm Description	125
4.6.2	Probability-Based Discovery	126
4.6.3	AN Algorithm Description	130
4.6.4	Anycast-based Discovery	138
4.6.5	SN Algorithm Description	140
4.6.6	Multi-dimensional Discovery	141
4.6.7	Query Processing Example	143
4.7	Resource Management	144

4.7.1	Requirements	146
4.7.2	ElCore Architecture	147
4.7.3	Resource Management Component	149
4.7.4	Resource Management System Operation	152
4.7.5	Resource Management for FMCS	155
4.8	Conclusion	160
5	Evaluation	163
5.1	Introduction	163
5.2	Simulation Environments	164
5.2.1	SIMNOW and COTSON	164
5.2.2	OMNET++	165
5.3	HARD Evaluation in Small Scale Systems	165
5.4	HARD Evaluation in Medium and Large Scale Systems	167
5.4.1	Scalability for Synchronous Querying	167
5.4.2	Heterogeneity and Complexity	171
5.4.3	Scalability for Asynchronous Querying	176
5.4.4	Comparison with Different Proposals	181
5.4.5	Other Features	187
5.5	ElCore Evaluation	189
5.5.1	Simulation Setup	190
5.5.2	Resource Allocation Accuracy (Mapping Quality)	193
5.5.3	Scalability and Performance	194
6	Concluding Remarks	197
6.1	Summary of the Thesis	197
6.2	Future Research Directions	199
	References	201
	Appendix	237
A	Implementation of HARD3 Modules in OMNET++	238
A.1	Simulation Process in OMNET++	239
A.2	HARD3 Components	239
	Index	243

List of Acronyms

ACO	Ant Colony Optimization.
AI	Arithmetic Intelligence.
ALU	Arithmetic Logic Unit.
AN	Aggregate-Node.
ANN	Artificial Neuron Network.
ASL	Average Search Length.
ATNoG	Advanced Telecommunications and Networks Group.
AVL	Georgy Adelson-Velsky and Evgenii Landis.
BCA	Bee Colony Algorithm.
BF	Bloom Filter.
BFS	Breadth First Search.
BOINC	Berkeley Open Infrastructure for Network Computing.
BRW2	a 2-layered hybrid broad-cast and full random-walk based discovery.
Cell BE	Cell Broadband Engine.
CP	Client Proxy.
CPU	Central Processing Unit.
DAML	DARPA’s Agent Markup Language.
DFS	Depth First Search.
DHT	Distributed Hash Table.
DIS	DHT Information Service.
DNS	Domain Name System.
DOS	Distributed Operation System.
DoS	Denial of Service.
DPT	Distributed Probability Table.
ElCore	Elastic Core.
FALKON	Fast and Light-Weight Task Execution Framework.
FMCS	Future ManyCore Systems.
FPGA	Field Programmable Gate Array.
FRW2	a 2-layered hybrid DHT and full random-walk based discovery.
FTR	Frequency of the Target Resources.
GFLOPS	Giga Floating-point Operations Per Second.

GIIS	Grid Index Information Service.
GIS	Grid Information Service.
GMD	Grid Market Directory.
GPU	Graphics Processing Unit.
GRIS	Grid Resource Information Service.
HARD	Hybrid Adaptive Resource Discovery.
HARD2	a 2-layered instantiation of HARD.
HARD3	a 3-layered instantiation of HARD.
HDL	Hardware Description Language.
HPC	High Performance Computing.
HTC	High Throughput Computing.
JCS	Jungle Computing System.
LDCE	Large Scale Distributed Computing Environment.
LLH	Length-Limited Huffman.
LN	Leaf-Node.
MAS	Multi Agent System.
MBFS	Modified Breadth First Search.
MDS	Monitoring and Discovery System.
MIPS	Millions of Instructions Per Second.
MTC	Many Task Computing.
NED	Network Description.
NIC	Network Interface Controller.
NoC	Network on Chip.
OIL	Ontology Inference Layer.
OS	Operating System.
P2P	Peer-to-Peer.
PA	Primary Attribute.
PM	Process Manager.
PRW2	a 2-layered hybrid DHT, learning-based and partial random-walk based discovery.
QMS	Query Management Service.
QoS	Quality of Service.
QREG	Query Registry.
QROUT	Query Router.
RAM	Random Access Memory.
RCaT	Resource Category Tree.
RCT	Resource Cost Table.
RD	Resource Discovery.
RDF	Resource Description Framework.
RDP	Resource Discovery Protocol.
RM	Resource Manager.

RMC	Resource Manager Component.
RP	Resource Information Provider.
RP-QMS	Resource Information Provider of type Query Management Service.
RP-SQMS	Resource Information Provider of type Super Query Management Service.
RR	Resource Requester.
S(o)OS	Service oriented Operating System.
SIMD	Single Instruction, Multiple Data.
SISD	Single Instruction, Single Data.
SLA	Service Level Agreements.
SMP	Symmetric Multi Processing.
SN	Super-Node.
SOA	Service Oriented Architecture.
SOC	Self Organized Cloud.
SON	Semantic Overlay Network.
SoOSim	S(o)OS Simulator.
SoR	Source of Resource.
SP	Server Proxy.
SQMS	Super Query Management Service.
TDD	Transmitted Discovery Data.
TORQUE	Tera-scale Open-source Resource and Queue Management.
TTL	Time To Live.
UDDI	Universal Description, Discovery and Integration.
VEE	Virtual Execution Environment.
VEEH	Virtual Execution Environment Host.
VEEM	Virtual Execution Environment Management System.
VM	Virtual Machine.
vnode	virtual node.
VO	Virtual Organization.
WRMS	Workload and Resource Management System.
WSDL	Web Services Description Language.
XDR	External Data Representation.
XML	Extensible Markup Language.

List of Figures

1.1	Thesis Organization.	6
2.1	Classification of the relevant resource discovery aspects.	10
2.2	Centralized architecture.	12
2.3	Decentralized architecture.	12
2.4	Hierarchical architecture.	13
2.5	Decentralized-Tree architecture.	13
2.6	Large scale computing environments.	14
2.7	Examples of query processing in Chord.	19
2.8	Routing example in Pastry: Query with the key 322323 arrives to the node 322311 which has the closest node-id. The routing algorithm corrects one digit at each step and then uses "leaf-set" to locate node with closest node-id to target.	21
2.9	Example of partitioning in 2-Dimensional CAN. Each node is assigned a unique zone in the M -Dimensional coordinate space. A new node is usually joined by identifying and splitting a zone in the coordinate space that can be split.	22
2.10	Sample routing path from S to D in a 2-Dimensional CAN. Using the greedy routing strategy, the query message, in each intermediate node, is routed to a neighboring node that is positioned closer to the desired location.	23
2.11	Computing strategy in Batch Systems.	30
2.12	Condor matchmaking.	30
2.13	Search algorithms for resource discovery.	40
2.14	Examples of query processing using BFS and MBFS.	41
2.15	Examples of query processing using Standard Random Walk and K-Walkers Random Walk.	42
2.16	Examples of information dissemination using ALG-Flooding and Swamping.	44
2.17	Examples of information dissemination using Random Pointer Jump and Name Dropper.	45
2.18	The important elements which have impact on the scalability of a resource discovery system.	53
3.1	An example of description of resources in hierarchical layers.	69
3.2	Distribution of resource information in a four-layers-hierarchy with 2 cells.	69
3.3	An example architecture of the hierarchical hardware description.	72
3.4	Fish eyes's model of distribution of the hardware capability information within the system.	74

3.5	(a) Configurable SIMD processing unit. (b) A simple dual core setup (bw: bandwidth; la: latency).	82
3.6	Aggregation procedure.	86
3.7	Attribute extraction procedure.	86
3.8	Binary string template for LLH-Hex encoding.	87
3.9	LLH encoding.	88
3.10	Combination of Huffman encoding and Run-Length encoding.	89
4.1	HARD3 overall system architecture.	97
4.2	Examples of cluster shapes (for two systems with different sizes and random network topologies) before and after applying the self-organization algorithm.	100
4.3	Overlay construction : Stage 1.	102
4.4	Examples of various situations of message processing for overlay making (with maximum group size=3)	103
4.5	Overlay construction considering Stage 2 to Stage 4.	105
4.6	An example of merging a source group into a destination group, suggested by a spot node	107
4.7	Some examples of interaction between distributed peers.	108
4.8	Evaluation results for the Load distribution in different network sizes (average number of processed messages per node for overlay making).	110
4.9	Evaluation results for the Load distribution among the nodes for network size=600 nodes and network size=1000 nodes.	111
4.10	Evaluation results for the clustering efficiency for different locality factors and network sizes.	112
4.11	Types of nodes in our hierarchical structure.	114
4.12	Transaction of the query messages between layers.	114
4.13	Initialization phase.	118
4.14	Resource requester general behavior.	120
4.15	Communication events within layer and across layers.	123
4.16	Mechanism of resource discovery according to the cost of resources (the numbers indicate the sequence of each message).	127
4.17	RCT- Assessment resources cost per type of query, for various ranges of latency.	127
4.18	Selection of the consequence QMS based on values of probe factors and probabilities.	129
4.19	Main query processing in QMS_{or} .	131
4.20	Examples of sequence of the resource discovery message flow for different sub-query schemes (the Main-Query contains a single sub-query).	132
4.21	An example of DPT in a system with 2 predefined attributes over 4 different ranges of values (resource-types or resource-categories) (CCR: Core Clock Rate, L1S: L1 Cache Size).	136
4.22	An example of sequence of the resource discovery message flow for simple querying (i.e., single-resource/single-query: the main-query contains a single sub-query desiring a single resource with particular query conditions in different layers).	143
4.23	ElCore resource management architecture	148
4.24	a) Extraction of the resource requirements from a given application graph, b) The resource graph of the qualified resources for a given query.	150

4.25	RMC mechanism.	151
4.26	Message sequences for the interactions between RR, RP and RMC.	151
4.27	Interaction diagram among modules.	153
4.28	Example of resource contention occurs when two (or multiple) different RMC instances (like RMC1 and RMC2 on behalf of PMi and PMj) simultaneously attempt to obtain the same or overlapping set of rare resources (like Rx).	155
4.29	Example of vnode clustering for a processor containing 24 heterogeneous cores, connected through 2-D mesh topology	157
4.30	Example of matching strategy for inter-resource communication requirements using Pivot Switching mechanism. A sub-query, for 5 homogeneous resources with given maximum inter-resource latency, initiated from RP-QMS _{org} and explores vnodes (from the closest vnodes to the most far one). The sub-query switches the pivot, when the recent matched resource fails to meet the upper bound communication requirement (maximum latency).	159
5.1	Overview of the evaluation strategy.	164
5.2	Average discovery latency for single requester, 25% and 40% of nodes as requesters with interval 30s.	166
5.3	Average discovery overhead in different network size for 1 requester and 50 % of nodes as requesters with query rate=.033/s.	167
5.4	Average number of discovery messages vs system size for basic querying (i.e., single-resource/single-sub-query) in several querying iterations (time periods) with different FTR values. FTR is the overall frequency of the target (desired) resources.	170
5.5	Average discovery latency vs system size for basic querying (i.e., single-resource/single-sub-query) in several querying iterations (time periods) with different FTR values. FTR is the overall frequency of the target (desired) resources. a) FTR=30, b) FTR=60, c) FTR=90, d) FTR=120.	171
5.6	a) Discovery overhead (in terms of number of required discovery messages) vs system size for different frequencies of the target resources in synchronous querying, b) Discovery latency vs system size for different frequencies of the target resources in synchronous querying.	172
5.7	Complexity (in terms of heterogeneity of desired resources and number of desired homogeneous resources per sub-query) vs discovery latency and number of discovery messages for different system sizes and different distribution of SoRs capabilities: a1 and a2) uniform SoRs with system size=3000, b1 and b2) uniform SoRs with system size=5000.	174
5.8	Complexity (in terms of heterogeneity of desired resources and number of desired homogeneous resources per sub-query) vs discovery latency and number of discovery messages for different system sizes and different distribution of SoRs capabilities: c1 and c2) uniform SoRs with system size=7000, d1 and d2) non-uniform SoRs with system size=16500.	175
5.9	Complexity (in terms of heterogeneity of desired resources and number of desired homogeneous resources per sub-query) vs discovery latency and number of discovery messages for different system sizes and different distribution of SoRs capabilities: e1 and e2) non-uniform SoRs with system size=27500, f1 and f2) non-uniform SoRs with system size=38500.	176

5.10	An overview of the results: complexity vs discovery latency and number of discovery messages for different system sizes and different distribution of SORs capabilities.	177
5.11	Density scatter plot of discovery cost (i.e., number of transacted discovery messages) for fully resolved discovery requests (i.e., hit rate=100%) over time for various application size (i.e., application execution time) while the querying interval is [2000,6000] ms: a1) execution time=(0,2000] ms, a2) execution time=(2000,4000] ms, a3) execution time=(4000,6000] ms, a4) execution time=(6000,8000] ms.	179
5.12	Density scatter plot of discovery latency for fully resolved discovery requests (i.e., hit rate=100%) over time for various application size (i.e., application execution time) while the querying interval is [2000,6000] ms: b1) execution time=(0,2000] ms, b2) execution time=(2000,4000] ms, b3) execution time=(4000,6000] ms, b4) execution time=(6000,8000] ms.	180
5.13	Mean hit rate for queries within the time-frames (each time-frame=10000 ms) for different application execution times in asynchronous querying with system-size=27500 resources.	181
5.14	Comparison between HARD3 and other alternative approaches: a, b & c) average number of required discovery messages and discovery latency per discovery request for different strategies and system sizes.	184
5.15	Comparison between HARD3 and other alternative approaches: a1 & a2) mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system sizes for HARD3, b1 & b2) mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system sizes for PRW2, c1 & c2) mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system sizes for FRW2.	186
5.16	Comparison between HARD3 and other alternative approaches: a1) average discovery load (number of transmitted discovery messages) per node (i.e., <i>vnode</i>) during simulation time (60000-80000 ms) for various strategies and system sizes, a2) provides better resolution of the results presented in a1 for different strategies excluding BRW2, b) average overlay load (number of transmitted messages for overlay construction) per node during simulation time (60000-80000 ms) for various strategies and system sizes, c) average transited discovery data per node during simulation time (60000-80000 ms) for various strategies and system sizes.	188
5.17	Manycore Simulation - System Architecture	191
5.18	Dependency graph for different settings	192
5.19	RAA ratio for different settings and network topology parameters: a) Setting 1: the processes with 2 threads b) Setting 2: the processes with 3 threads c) Setting 3: the processes with 5 threads d) Setting 4: the processes with 9 threads (MRA is the Maximum Reachable Accuracy, each run indicates different random network topology parameters).	193
5.20	Resource Allocation Inaccuracy ratio for the processes with different number of threads and threads' dependency.	194
5.21	Hexbin plots for the number of transaction messages and the latency for each resource request over time.	195

5.22 a and b) Histogram, density plot and CDF plot for the number of transaction messages between resource management components over simulation time, c and d) Histogram, density plot and CDF plot for the querying latency for the PM requesters over simulation time	196
A.1 Overview of the Simulation Process in OMNET++	239
A.2 Class Hierarchy for HARD3 Module Development in OMNET++ (Simulation Component of HARD3).	240

List of Tables

2.1	Examples of resource discovery in Grid-based systems (DS: Discovery System).	16
2.2	Examples of resource discovery in P2P-based systems (DS: Discovery System, FoCs: Flocks of Condors, MaT: MatchTree, CyG: CycloidGrid, SkipC: SkipCluster).	18
2.3	Summary of comparison of the major types of resource discovery for different performance factors (PFs: Performance Factors, RB: Reliability, FT: Fault-Tolerance, LB: Load-Balance, AN: Autonomy, CQ: Complex Querying).	27
2.4	Examples of clustering approaches for resource discovery (OA: Overlay Architecture).	37
2.5	Comparison of some well-known synchronous search algorithms for state discovery.	43
2.6	Summary of bio-inspired search methods (SA: Search Algorithm).	47
2.7	Overview of the evaluation methods for resource discovery.	52
2.8	Some examples of the evaluation of scalability methods in other resource discovery research works (RD: Resource Discovery, ET: Evaluation Tools, PL: Planet Lab, BM: Berkeley Millennium, IF: Iamnitchi & Foster, SG: StarGrid, MT: MatchTree, OS: OntoSum, NA: Not Available).	55
2.9	Summary of comparison between some of the well-known resource discovery approaches for different evaluation factors.	61
2.10	Summary of comparison between some of the well-known resource discovery approaches for different evaluation factors.	62
2.11	Comparison between some of the well-known resource discovery approaches for different evaluation factors.	62
3.1	An example of layer definition with multi-dimensional attributes.	71
3.2	Examples of resource attributes in each layers.	73
3.3	An example description of general system attributes as well as the capability information for a sample single resource.	87
3.4	Information encoding	89
3.5	Information encoding - Summary	90
3.6	Cost of encoding (required space): NOS=Number of Fixed-Length Symbols, BPS=Required Bits per Symbol, BFL=Required Bits for the Fixed-Length, MCL=Maximum Permitted Codeword-Length, MBC=Maximum Required Bits for the Codewords, LID=Length of Sample Input Data by Bits, TCB=Total Cost or Maximum Cost by Bits.	90
4.1	Summary of the communication events for the overlay construction.	101

4.2	Input parameter values used in the experiments (Scenario 1).	109
4.3	Input parameter values used in the experiments (Scenario 2).	111
4.4	Inter-layer and intra-layer communication events.	124
5.1	Simulation parameters for scalability evaluation (synchronous querying).	168
5.2	Simulation parameters for HARD3 evaluation under the complex querying conditions and high heterogeneity of resources (synchronous querying).	173
5.3	Simulation parameters for asynchronous querying.	178
5.4	Simulation parameters for performance compression.	183
5.5	An overall comparison between HARD3 and examples of other discovery approaches in terms of querying features.	189
5.6	Simulation Parameters for Accuracy Evaluation	192
5.7	Simulation Parameters for Scalability Evaluation	194

List of Algorithms

1	WakeUp,JRequest.	103
2	JAccept.	104
3	Merging optimization.	106
4	Resource discovery general procedure.	116
5	Processing a Lookup sub-query by a RP_{ln}	125
6	DPT Updating	133
7	Processing an Update _{qms} sub-query by a RP_{an}	135
8	Selection of the next QMS provider	137
9	Anycast based Forwarding in RP_{sn}	141
10	Pseudo code of mechanism for resource requesting in RMC when PM asks RMC to provide the resource descriptors for a set of required resources.	152

Chapter 1

Introduction

Large scale distributed computing technologies such as Cloud, Grid, Cluster and High Performance Computing (HPC) supercomputers are progressing along with the revolutionary emergence of many-core designs (e.g. GPU, CPUs on single die, supercomputers on chip, etc.) and significant advances in networking and interconnect solutions [1]. In order to efficiently benefit from all the potential resources in such large scale many-core-enabled computing environments, Resource Discovery Protocols (RDPs) are the vital building blocks to maximally exploit the capabilities of all distributed heterogeneous resources, through precisely recognizing and locating those resources in the system.

1.1 Background and Context

Processors will not be able to enhance their (single-thread) performance exponentially [2]. Rather, due to the many-core nature of future large scale computing platforms, the evolution will proceed by scaling the number of processing cores. Consequently, application software will no longer get faster execution speeds automatically with each hardware upgrade, but will have to be adapted to get exposed to the higher level of parallelism by the Central Processing Unit (CPU). This means that to benefit from the many-core hardware improvements, we should reconsider our traditional concepts of applications, operating systems and compilers towards the direction of massively distributed, heterogeneous and dynamic computing environments. We can imagine that computing nodes in future HPC systems, with thousands of cores, may be connected together to form a single transparent computing unit, thus hiding the complexity and distributed nature of the many-core system from applications while it is expected that the computational systems extend far beyond the chip, with hundreds and thousands of chips incorporated into collaborative execution units in a wide-scale distributed structure.

Moreover, due to the specific requirements and limitations of the current HPC systems, particularly in terms of high dynamicity and high heterogeneity (e.g. the static configuration of the task execution environment which usually depends on specific applications, libraries, job schedulers and operating systems, restricts the service users to implement certain application scenarios) and also because of the steady progress of Service-oriented and Cloud computing paradigms, we can envision that the long-term future of HPC clusters will tend towards large scale distributed cloud-like systems supporting high performance computing with increasing flexibility and efficiency.

The emergence of large-scale computing technologies (such as Cloud, Grid, Cluster and

HPC) has led to increase complexity for integration of such diverse computing environments, infrastructures, platforms and technologies. Additionally, data distribution, hardware availability, software heterogeneity, and also the sheer size of scientific problems, commonly force scientists to use the benefit of Jungle Computing (i.e. leveraging multiple computing platforms simultaneously) instead of traditional supercomputers and clusters. Jungle Computing has emerged as a new modern distributed computing paradigm based on simultaneous combination of various hierarchical and distributed computing environments which are saturated with large number of heterogeneous resources [1, 3].

In fact, integration of resources would be necessary specially when no single resource is available that its computation capacity can meet the computation requirements, or if different parts of the computation have various computational requirements. Furthermore, for large number of users and applications, combination of multiple computing environments with various types of resources leads to achieve high peak performance by potentially accessing diverse collection of available resources while it is cost efficient since there exist many types of computing landscapes (i.e. isolated computing infrastructures) that can be efficiently integrated as so-called Compute Jungles. However, high heterogeneity (in terms of hardware, resources, connectivities and platforms) and high hierarchical design of Compute Jungles make them difficult and more complex for scientists to obtain an efficient use of these systems.

For distributed operating systems executed on such large scale many-core-enabled computing environments, Resource Discovery (RD) is a vital building block to maximally exploit the capabilities of all distributed heterogeneous resources. In fact, when we have a pool of variable-type and large number of processors, resource sharing becomes complex. This is specially true if we are trying to potentially migrate some applications to other (possibly different) processors in the system for overloaded processors. But before resource (re)allocation and execution migration, we need to match execution requirements to resources and locate them.

1.2 Motivation

In the future, computing nodes with thousands of cores may be connected together to form a single transparent computing unit which hides from applications the complexity and distributed nature of these many core systems. In such many core environments resource discovery is an important powerful building block to exploit the capabilities of all distributed resources. It aims to match the application's demands to the existing resources by discovering and finding resources.

The resource discovery functionality is required by advanced "Code-Analysis and Segmentation" to find the available different processing units and exploit their specific capabilities. It also uses some sort of "Resource Description Provider" to get information about the resources and employs "Resource Requirement Identifier" to get information about code specific resource requirements.

From the collected state-of-the-art and elaborated synthesis, a set of research objectives has been made in order to provide a path for handling concrete research questions. These objectives have been further enhanced with the consideration of integration with the activities in the host research institution, the Instituto de Telecomunicações (Aveiro Site), in particular the Advanced Telecommunications and Networks Group (ATNoG). Also, integration with international project, like Service oriented Operating System (S(o)OS) was one of the strong

development lines.

S(o)OS [4, 5] was a European project aiming to generate a reference architecture for future large scale, distributed infrastructures, considering the current trends in processor manufacturing including many-core systems, the memory wall problem, the growing degree of heterogeneity, high-dynamicity of the resources and the different hierarchies within the communication infrastructure.

1.3 Problem Statement

One of the most challenging issues in large scale computing environments (e.g. Cloud, Grid, HPC and Cluster) is scalable efficient resource discovery. This would be even more critical for Jungle Computing System (JCS) due to the specific requirements of these computing environments such as high heterogeneity, high hierarchy and high dynamicity . Resource discovery in JCS can be defined as a highly adaptive approach (considering the diversity of hardware, platforms and computing infrastructures) to instantaneously find the most appropriate available set of resources (i.e. computing resources like as processing cores) for the user applications in the system with minimum cost of communication and computation (i.e. in terms of network latency, network traffic, discovery load, etc.), based on the specific set of computational and communicational requirements for each application or application segments, in a way that the static and dynamic properties of resources as well as their interconnected and aggregated characteristics could be qualified according to the query requirements. Moreover due to the intimate complexity of JCS environments, high performance execution of applications requires that resource discovery supports some other important features such as proximity-awareness (i.e. the , the discovered resources must be close as much as possible) , querying flexibility (in terms of complex querying such as multidimensional querying).

Furthermore, resource discovery challenges for the next generations of many-core systems can be considered mainly as associated with scalability and efficiency issues. The description of arbitrary resources for a huge number of heterogeneous resources in an adequate and efficient manner is not practically attainable at the present time. All the diverse resources in a distributed system need to be defined by a set of strict parameters that adequately describe the characteristics and performance factors of the correspondent resources sufficiently. However, exploring all the existing cores, when different architectures and features are bound, and discovering the ones apt for a certain set of conditions on the local chip or on a larger scale network is almost unfeasible due to potential excessive information exchange. Therefore discovery mechanisms should be efficient enough to support large scale distributed heterogeneous (many-cores, and not only multi-cores) environments. Furthermore, with the potential high dynamicity of resources, providing an intelligent real-time resource awareness mechanism for managing and scheduling resources (cores), which is able to estimate an optimum resource allocation for a specified request, is a major challenge.

In fact, distributed operating systems for heterogeneous multi-core and many-core environments require efficient resource discovery methods to announce and discover processing resources around the network. These methods must have enough consistency and adaptation to work with on chip and off chip networks, Furthermore, the multiplicity of characteristics of the heterogeneous processing components such as interconnection networks, memory bandwidth, latency, cache size, etc., must be taken into account for designing a model for resource description and resource discovery.

We must also note that values such as clock rate, Millions of Instructions Per Second (MIPS), Giga Floating-point Operations Per Second (GFLOPS), cache size, etc., are useful metrics to describe computing units, but they are not reliable enough and applicable to operate as complete resource descriptions [6]. Micro Benchmarks such as HPL, NAMD, or SPEC [7] attempted to solve this problem from the application side, testing a set of standard algorithms on the target system. However, there is still not an entirely accurate and precise performance metric to properly describe resources in a many-core system in general. The problem of resource identification and discovery, respectively can be structured along the following two questions:

1. How to identify the resources required for a given process? or how to identify the hardware resources requirements from either the static (source code) or the dynamic (runtime) behavior of the program (i.e., Resource Identification)?
2. How to find the required resources for a given query (i.e., Resource Discovery)?

For running a given process optimally, we need to identify the resources which maximize the matching between resource capabilities and process characteristics. This could be done by using a code analyzer component (based on methods such as meta data provided by developer) or examining the characteristics and behaviors which are exhibited by the code itself. For example, in the case of a process with several data inputs and single instruction, perhaps the best matching resource could be a Single Instruction, Multiple Data (SIMD) vector processor. In the other example, for a given parallel application containing several threads, according to its dependency graph, the required resources could be a set of processing cores which are connected to each other with the links that satisfy the specific communication requirements. Upon identification of the adequate resources for a given process, the user or any responsible Operating System (OS) component can generate a query for discovery. Afterwards, the resource discovery module would be responsible to find the best matching resources for the query through efficient exploration of the resources in the whole system.

In this work we are proposing Hybrid Adaptive Resource Discovery (HARD), a novel efficient and highly scalable resource-discovery approach which deals with the aforementioned resource discovery challenges while it is applicable to large heterogeneous and highly dynamic distributed computing environment like JCS.

1.4 Research Aims and Objectives

This research will address resource discovery problem in future multi-core and many-core systems, especially for distributed and parallel operating systems. The main aim of the research is to enhance the usage efficiency of all available processing resources in an environment with huge number of homogeneous and heterogeneous cores. To achieve this target, minor objectives are:

- To design and develop a new resource discovery approach for arbitrary-scale distributed computing environments.
- To provide scalability and adaptability for resource discovery in highly dynamic and highly heterogeneous computing environments.

- To reduce discovery load (in terms of number of transaction messages per query) while maintaining a global system view.
- To provide efficiency in terms of discovery latency for resource discovery in arbitrary-scale systems.
- To provide high flexibility for complex resource discovery through supporting features such as multidimensional querying, resource graph querying, exact matching and partial matching.
- To provide accuracy for resource discovery in order to precisely recognize all the matched desired resources in the system for each query.
- To increase the rate of discovery consistency which it also means minimizing the rate of false service discovery (Consistency in service discovery protocol means that the discovered services will be matched with the actual services which are advertised by the origin nodes. While the size of a network becomes larger, the number of service entities also increases, however, the network parameters such as packet loss, core load, and transmission delay take longer to propagate. Therefore, the inconsistency or rate of false service discovery between the original services and discovered services is raised.)

1.5 Tools and Methodology

In this work, we use the C++ language and simulation tools for module development, implementation and assessing the performance of our proposed resource discovery approach. For doing this, we develop our resource discovery modules using C++ while we use HP COTSon [8], AMD SimNow [9], OMNET++ [10] and OverSim [11] simulators for environment simulation and evaluation purposes. We also use the R language and Matlab for result analysis. COTSon and SimNow are employed to precisely simulate the small and medium scale real world-like many-core-enabled computing environments while preserving a significant amount of computation details. In fact we prove the validity and functionality of our work through evaluations using COTSon and SimNow. We also leverage OMNET++ and OverSim to extend our evaluations to medium and large scale computing environment with more focus on communication parameters.

1.6 Thesis Organization

The organization of dissertation is shown in Figure 1.1. In Chapter 1, we outline the main motivation and objectives for investigating the problem of resource discovery in arbitrary-scale computing environments. Chapter 2 presents the current state of the art in the field which this work addresses. On one hand, networking and hardware technologies used in heterogeneous multi-core / many-core environments are reviewed. On the other, current resource discovery and management protocols, mechanisms and techniques are discussed. Also, a brief introduction is included about aspects of current distributed operating systems which are directly related to the contributions made in this thesis, mechanisms for discovery of service and for future distributed many-core environments.

Chapter 3 starts by investigating the hardware description and modeling issues for distributed operating systems with regards to the many-core high performance computing

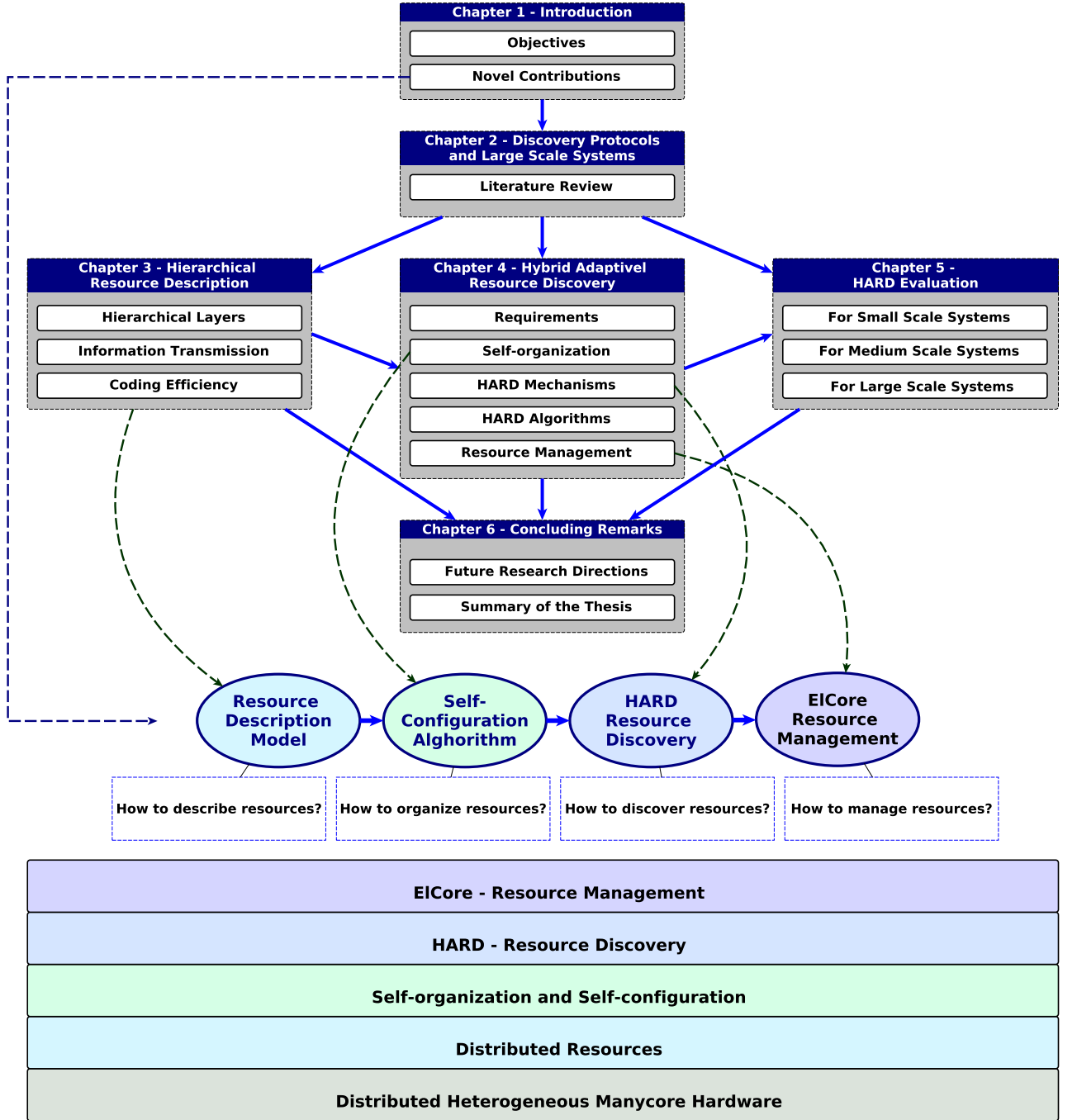


Figure 1.1: Thesis Organization.

environment and presents our hierarchical resource description model which is used for resource discovery purpose. In Chapter 4 we propose a new efficient and scalable resource discovery protocol, called HARD for arbitrary-scale many-core-enabled computing environments. We also propose a multi-step, load-balanced, self-organized, clustering algorithm for

overlay establishment (particularly for resource discovery) in distributed environments which preserve the locality of the nodes within the groups while it deals with efficiency and scalability. In the last section of this chapter we also present a new resource management model for distributed computing systems which is based on the proposed resource discovery protocol.

In Chapter 5, we present the experimental results to evaluate and analyze the performance of the proposed resource discovery protocol under several different situations and simulation scenarios. Finally, Chapter 6 summarizes the main conclusions of the dissertation and contains the contributions of the thesis by presenting an overview of individual contributions per chapter and also our suggestions for future research.

1.7 Novel Contributions

The novel contributions of the thesis include the following:

- New efficient, highly scalable resource discovery protocol for many-core-enabled arbitrary scale computing environments which is built upon a virtual hierarchical overlay based on self-organization and self-adaptation of processing resources in the system where the computing resources are organized into distributed hierarchies according to a proposed hierarchical resource description model (i.e. multi-layered resource description). The proposed approach supports distributed query processing and aggregation of discovery results within and across hierarchical-layers by leveraging various distributed resource discovery services and functionalities in the system, implemented using different adapted algorithms and mechanisms in each level of hierarchy. It also supports complex querying (e.g. multidimensional querying, resource graph querying, etc.,) for highly heterogeneous and hierarchical dynamic computing environments.
- New scalable hierarchical resource description and hardware modeling which is able to abstract the characteristics and behavior of the large scale many-core-enabled computing system in a way that both computational and communicational system properties are well represented to provide a close estimation of the real system while avoiding to describe the hardware at the cycle-accurate level. This model is far too detailed and general for task management and resource discovery.
- New generic resource management scheme which can be applied and adjusted to the requirements of different future complex computing distributed environments through by focusing on resource allocation issues. It provides a high level of accuracy for resource allocation which has a significant impact on the overall system performance.
- New scalable, load balanced, multi-stage hierarchical clustering algorithm which automates the process of the optimally composing communicating groups in a dynamic way while preserving the proximity of the nodes.

The aforementioned contributions resulted in the following list of scientific publications:

- Javad Zarrin, Rui L. Aguiar, João Paulo Barraca, Dynamic, scalable and flexible resource discovery for large-dimension many-core systems, *Future Generation Computer Systems*, Volume 53, December 2015, Pages 119-129, ISSN 0167-739X, <http://dx.doi.org/10.1016/j.future.2014.12.011>.

- Javad Zarrin, Rui L. Aguiar, João Paulo Barraca, ElCore: Dynamic elastic resource management and discovery for future large-scale manycore enabled distributed systems, *Microprocessors and Microsystems*, Available online 22 June 2016, ISSN 0141-9331, <http://dx.doi.org/10.1016/j.micpro.2016.06.007>.
- Javad Zarrin, Rui L. Aguiar, and Joao Paulo Barraca. A self-organizing and self-configuration algorithm for resource management in service-oriented systems. In 19th IEEE Symposium on Computers and Communications (IEEE ISCC 2014), Madeira, Portugal, June 2014.
- Javad Zarrin, Rui L. Aguiar, João Paulo Barraca, A Specification-based Anycast Scheme for Scalable Resource Discovery in Distributed Systems, *Proc. 10th ConfTele 2015 - Conference on Telecommunications*, Aveiro, Portugal, Sep 2015

and the following manuscripts in submission:

- Javad Zarrin, Rui L. Aguiar, João Paulo Barraca, HARD: Hybrid Adaptive Resource Discovery for Jungle Computing, *Journal of Network and Computer Applications (JNCA)*, Elsevier, February 2016. (First Revision Submitted)
- Javad Zarrin, Rui L. Aguiar, João Paulo Barraca, Manycore Simulation for Peta-scale System Design: Motivation, Tools, Challenges and Prospects, *Journal of Simulation Modelling Practice and Theory (SIMPAT)*, Elsevier, July 2016. (First Revision Submitted)
- Javad Zarrin, Rui L. Aguiar, João Paulo Barraca, Resource Discovery for Distributed Computing Systems: A Comprehensive Survey, *Journal of Parallel and Distributed Computing (JPDC)*, Elsevier, October 2016.

Chapter 2

Discovery Protocols and Large Scale Systems

Large scale distributed computing environments provide a vast amount of heterogeneous computing resources from different sources for resource sharing and distributed computing. Discovering appropriate resources in such environments is a challenge which involves several different subjects. In this chapter, we provide an investigation on the current state of resource discovery protocols, mechanisms and platforms for large scale distributed environments, focusing on the design aspects. We classify all related aspects, general steps and requirements to construct a novel resource discovery solution in three categories consisting of structures, methods and issues. Accordingly, we review the literature analyzing different aspects for each category. We also review current literature describing approaches for resource management for different distributed computing systems at the end of this chapter.

2.1 Introduction

In recent years, large scale heterogeneous computing infrastructures such as Grids [12], Clusters [13], Clouds [12], Hybrid HPCs [13], Peer-to-Peer (P2P) environments or even simultaneous combination of multiple platforms have become increasingly wide spread due to the development of many-core technologies and associated advances in high performance and distributed computing. Despite the fact that these platforms provide distinct computing environments, they have a fundamental common key property; the ability to share resources/services belonging to different administrative domains among all the entities distributed across the whole system. Resource sharing produces significant benefits exploring the idle cycles of potential available resources over the system by integrating, leveraging and utilizing the potential of these myriads of individual available resources to achieve higher computing capabilities. The novel challenges arise from the fact that it would not be feasible to statically maintain the global knowledge of all the available distributed sharing resources for the existing entities in a dynamic Large Scale Distributed Computing Environment (LDCE). Indeed, one of the most challenging essential issues in such environments is the problem of resource discovery where each entity in the system has to be able to potentially explore and involve the desired resources to attain the relegated computations and services.

Resource discovery encompasses locating, retrieving and advertising resource information, being an essential building block to fully exploit all distributed resources in the system. The

purpose of resource discovery in a LDCE is to enable the adaption of the application's demands to the resources potentials by discovering and finding resources with deep understanding of the resource specifications according to application's resource requirements. However, considering the nature of LDCE, any approach for resource discovery in such environment needs to be qualified in some challenging issues such as scalability, efficiency, heterogeneity and reliability. As discussed in this chapter, there are considerable amounts of research works done in the area of resource discovery which study these challenges in different aspects pursuing assorted objectives. In this chapter, we provide an investigation on the discovery protocols and their relevant aspects specific for large scale computing. We describe all the potential related concepts and terminologies. Accordingly, we provide a taxonomy to categorize the various discovery systems and techniques while we analyze and compare different approaches. We also limit our study to discovery solutions which are applicable to be employed in LDCE as there are some more traditional surveys in previous discovery [14–20].

In this chapter, we define a resource as any source of supply [21]. In the specific case of large scale computing we limit the definition of resource to computing resources that means any element which is directly involved in computing process such as CPU-capabilities, Random Access Memory (RAM), disk space, communication and network-capabilities, etc.,.

2.2 Relevant Discovery Aspects

In-order to create a resource discovery solution, there are several fundamental open issues and involved elements which have significant impact on the discovery final behavior, operation, functionality and even the way that it can deal with particular challenging issues and objectives. In this chapter, we have categorized and explained all the major common concepts, structures, techniques and functionalities that shape the critical aspects of every resource discovery models in three classes, as described below (see Figure 2.1).

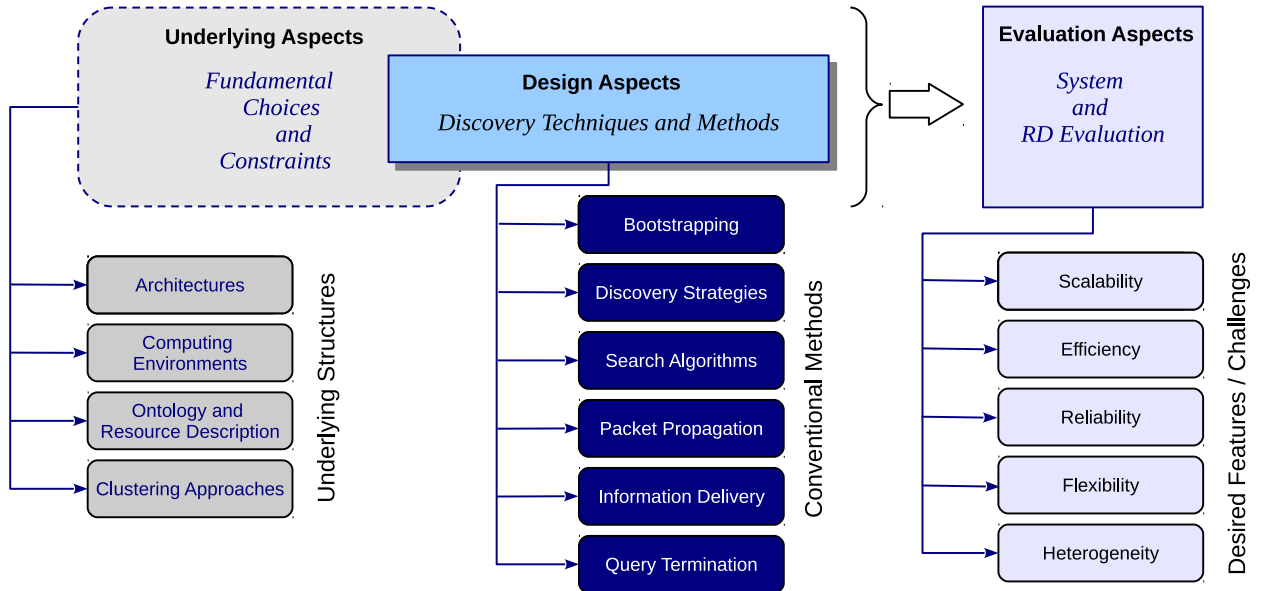


Figure 2.1: Classification of the relevant resource discovery aspects.

1. **Underlying Aspects** contain all the things that should be prepared and configured before discovery usage (e.g., stating a query). These aspects are more relevant to the underlying computing environment, network topology data infrastructure, representation of resources (resource abstraction), and resource information distribution.
2. **Design Aspects** include the current major possible relevant techniques, strategies and methods that can be used to guide queries between distributed entities in order to locate and discover resources. Search algorithms, mechanisms for packet propagation, querying strategies, resource information delivery, data synchronization between distributed resource providers and information summarization techniques are examples of such elements.
3. **Evaluation Aspects** cover all the concepts and terms that express, demonstrate or evaluate the expected or desired achievements of the resource discovery procedure. Depending on the specific environment, applications and objectives that a discovery protocol has been designed for, the number of certain functionalities, features and performance factors can become critically important to attain. For example, for the resource discovery in wireless sensor networks, energy efficiency is one of the critical performance factor while in the area of large scale computing, scalability has the key role.

In the continuation of this chapter, we describe and explain the aforementioned elements which play a key role in designing and operating of resource discovery protocols.

2.3 Underlying Aspects

2.3.1 Architectures For Resource Discovery

In the distributed system, before designing a resource discovery model it is necessary to examine the general distribution options in environments and clarify a discovery architecture depending on the application, adaptation to the target computing environment and set of other important involved parameters. From the state-of-the-art [14, 15, 22–26], we can categorize the discovery architectures in four distinguishable groups which are centralized, hierarchical, decentralized and hybrid (e.g., decentralized-tree and decentralized mesh).

In the centralized case (e.g., [27–31]) (Figure 2.2), the resource information about all other nodes and instances is located at a central point, that can be reached by all service or resource requester instances in the environment. All the resource providers periodically update and register their dynamic or static information in the central repository. The central information service is the only entity in the system who can process the queries from resource requester in order to match the query to the resources available. This has the advantage that the information source is always known and that reallocation of services / resources can be more easily propagated. At the same time, the central information service, becomes a bottleneck and a single point of failure. In fact, fully centralized systems generally do not scale well.

As opposed to the centralized architecture, in the decentralized case (e.g., [22, 32, 33]) (Figure 2.3), all (or, at least, several) nodes have to host at least the base information and capabilities related to query processing and management, so that they do not have to query a central instance for query analysis and get the information about the desired resources every time. Accordingly, the system becomes more failure resilient and in particular much more

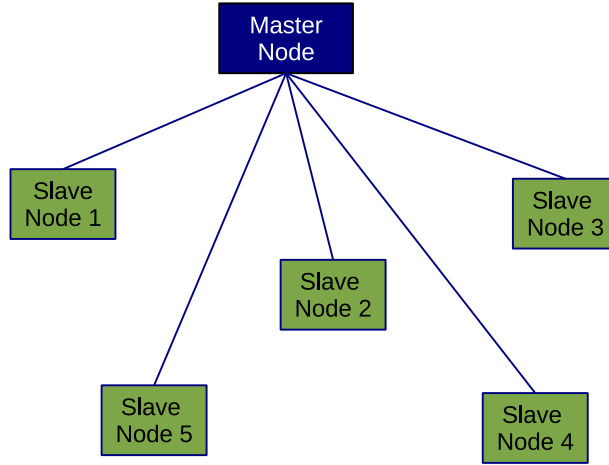


Figure 2.2: Centralized architecture.

scalable. However, depending on the use case, the degree of distribution can quickly lead to a management communication overhead if all nodes have to share the same information from different endpoints. In these cases the decentralized case leads to a all-to-all communication (e.g., query flooding) across the whole network. However, depending on the case, using some mechanisms (e.g., selective search and multi casting) we can reduce the overhead problem from all to all communication (broadcast), to one to many (multicast or anycast) or one to one (unicast) communication.

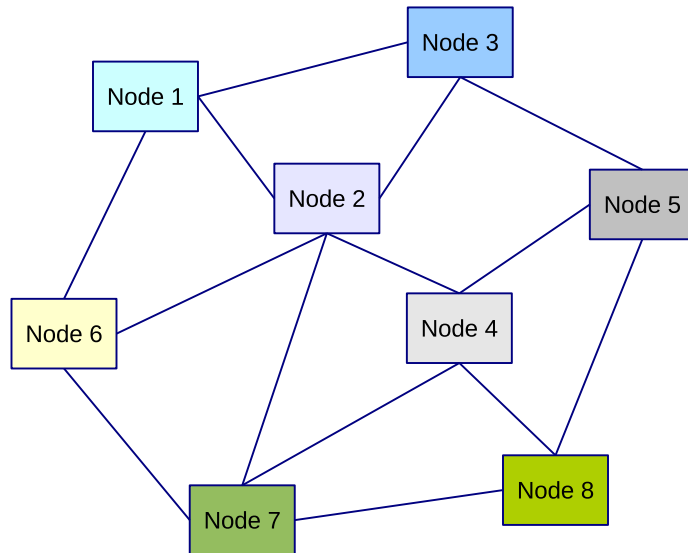


Figure 2.3: Decentralized architecture.

Nevertheless, it is always more difficult to keep track of potential query-resource reallocations (caching), as the resource information (e.g., resource location) either has to be distributed to all potential communication points, or some mechanisms for routing and updating has to

be introduced.

The hierarchical architecture (Figure 2.4) through which the relevant information is propagated in a hierarchical fashion, i.e., through locally connected layers that each in turn are connected to lower layers. This way, the information does not need to be propagated throughout the whole network and the communication overhead is restricted to the substructure. At the same time, however, propagating the layers adds delay due to the messaging route, so that lower level will get access to the information later than higher levels.

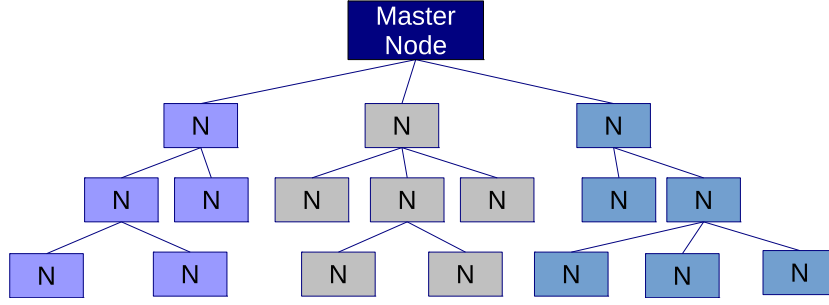


Figure 2.4: Hierarchical architecture.

These models may benefit from combination of features. Decentralized-tree (e.g., [34, 35]) (Figure 2.5) and decentralized-mesh (e.g., [36]) are types of hybrid architecture which have been designed to aggregate the advantages of the traditional architectures while reducing their shortcomings.

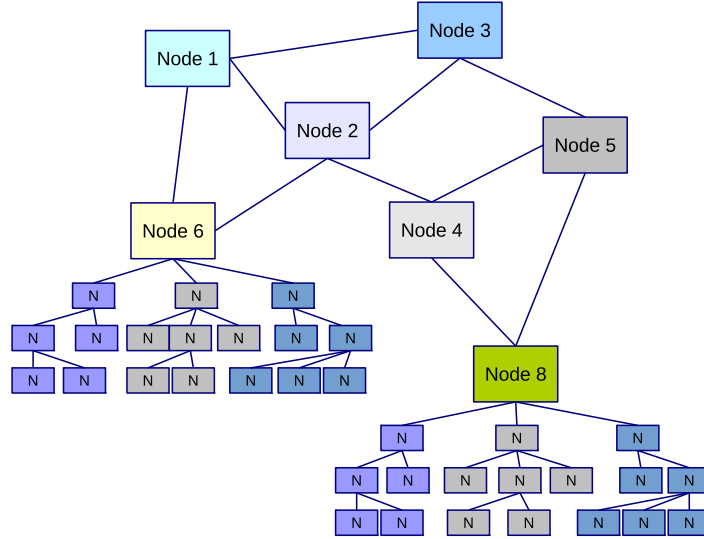


Figure 2.5: Decentralized-Tree architecture.

As practical example of this kind of architectures, we can mention the resource discovery in many core environment, where the individual resource description sources can encapsulate multiple levels of aggregated resources. This higher level information can either integrate all lower-level resource descriptions, or abstract from it by providing a more holistic view on the

information. Either choice affects the communication strategy as more information implies a higher amount of data (and according maintenance), whilst more abstract data implies that additional queries may have to be executed in order to acquire details that are potentially required.

2.3.2 Computing Environments

The computing environment has a large impact on the architecture, operation, implementation methods, and the performance of resource discovery protocols. In this section, we describe and classify common resource discovery approaches for the most important (and more conventional) large scale computing environments including Grid, P2P, Cloud and Cluster (see Figure 2.6).

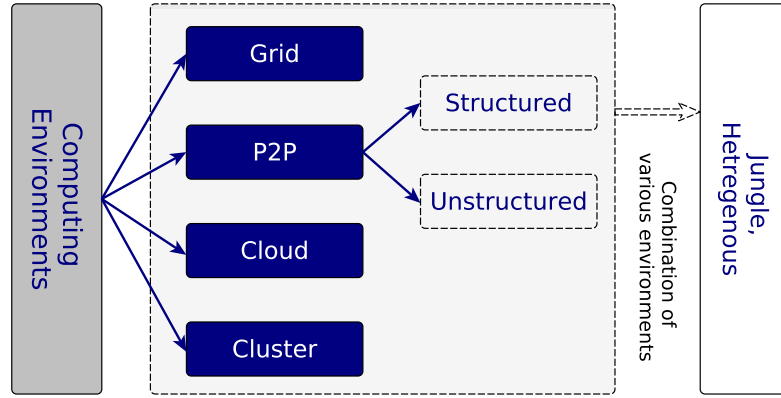


Figure 2.6: Large scale computing environments.

2.3.2.1 Grid Systems

Grid computing [12, 13] generally contains a large amount of heterogeneous resources which potentially are loosely coupled and geographically distributed. This “large group of resources” (devices, data, applications, services, storage, sensors, computational clusters, parallel super computers and any other sort of computing or communicating resources) act together as a single super powerful computing system in-order to perform large tasks. The Grid originally has been designed to efficiently map the user’s jobs to the most appropriate free resources. In order to process user’s jobs it is essential to get resource information in a minimum time. There are various types of Grids depending on their specific objective and focus such as data, application, service, knowledge, interaction, utility and computing Grid. In this chapter, we mainly address on computing Grids.

Resource discovery in Grid systems aids in resource management and application scheduling and generally suffers from two major challenges: efficiency and complexity. First, the majority of the existing Grid Information Services (GISs) [37] (like Monitoring and Discovery System (MDS) [38]) will organize their resources based on non-adapted overlays where the categorization of resource characteristics and information doesn’t play any role in the overlay construction, being dependent on administrative issues. This leads to inefficient query forwarding across

the whole system while a resource-aware clustering overlay could reduce the general discovery overhead greatly by limiting the query traversing area. The second challenge is providing the capability for Grid discovery to perform complex querying. Grid job schedulers are responsible to allocate user's jobs to the most appropriate resources. Basic querying techniques like exact-matching are not enough to precisely map the jobs requirements to the resources specifications. The drawback is that the exact matching heavily depends on the exactness of the query's conditions and it requires to specify the exact jobs requirements, which might be not feasible in practice. In other words, exact matching strategies are not suitable for approximate matching when the exact requirements of the jobs are not clear. Moreover, exact matching ignores the resources which have specifications overqualified for the job's requirements. This leads to underutilization of the system's resources, causing the overqualified resources to be idle, when exact match fails. Thus, leveraging multiple complex querying techniques such as keyword search, range querying, partial matching, multidimensional and resource graph querying becomes a necessity to perform adequate job/resource matching.

Resources in Grid are defined as a set of attributes. A query simply describes the specific desired values for some of attributes. A simple example of a query is like "processor frequency=1.7 GHz" where the discovery process locates all the processing resources in the system which have exactly a 1.7 GHz processor. However, complex range querying like "2.1 GHz > processor frequency > 1.7 GHz" or multidimensional querying like "A1=V1, A2=V2, V4 > A3 > V3" generally are not supported in Grid environments. We will discuss complex querying issues in detail later.

Multiple Grid-based resource discovery approaches have been proposed to enable a GIS. They can be categorized in centralized, hierarchical decentralized and hybrid systems according to their main approaches to the discovery problem. Table 2.1 presents examples of resource discovery in Grid systems.

A) Centralized-Grid: Centralized model is the simplest approach to create an information service, where it is constructed by establishing a centralized directory agent. The major advantage of this solution is the simplicity of finding all resource information on the central server, making the resource discovery latency low, and data coherence high. However, these approaches suffer from sub-optimal scalability and lower fault tolerance, mostly due to the centralized nature of the directories, as discussed previously in Section 2.3.1. Nimrod-G [39] and Condor-G [40] are the examples of Grid super-schedulers where they have employed a centralized Grid information services such as R-GMA [41–43], Hawkeye [43, 44] and Grid Market Directory (GMD) [45] to index their resource information.

B) Hierarchical-Grid: Another approach for discovery in Grids relies in hierarchically organized servers. In MDS [38, 47], the top index server answers requests either directly or by dispatching them to its child index servers. This approach has limited scalability, as requests trickle through the root server, which can easily become a bottleneck and consequently suffer from fault tolerance issues. Indeed, the loss of a node in the higher level of the architecture causes the loss of an entire sub-tree. Ganglia [51], MDS-3 [53] and MDS-4 [49, 50] are another examples of hierarchically designed GIS.

C) Decentralized-Grid: The concept of decentralized discovery systems in Grid is related with the idea of peer to peer computing wherein there is no central server in the system. P2P Grid consists of a group of Grid peers which share their resources in a purely decentralized manner. A Grid peer contains a set of local nodes where each node (computing machine) can manage several Grid resources. All the nodes potentially can play the role of server or client. Since the importance of all the nodes are equal, the system does not suffer

Table 2.1: Examples of resource discovery in Grid-based systems (DS: Discovery System).

DS	Base	Mechanism
GMD [46]	Centralized-Grid	Provides a web-based approach for managing resources in Grid systems. It consists of two main components: a portal manager and query web service. Grid users can share and register their resources in Extensible Markup Language (XML) format manually in the central resource information repository, using the portal manager. Clients are able to search for their required resources in the central database using the query web service. The central node is the only peer in the system which is able to gather and store resource information and perform the query processing, therefore it easily becomes a bottleneck. Like as any other centralized system it suffers from poor scalability. Furthermore, as users manually should update their resource information, GMD doesn't support dynamicity in all aspects (e.g., dynamic attributes, dynamic topology). However, it supports some complex querying feature such as multidimensional and range querying since it has a central database to store resource information which those kind of queries easily can be supported.
	Hierarchical-Grid	Designed to enhance the primary version of MDS in order to be adapted in a hierarchical environment. It has two main components, a configurable Grid Resource Information Service (GRIS) and a directory component called Grid Index Information Service (GIIS). They are organized in a hierarchy where the higher nodes host GIIS and the lower nodes host GRIS. The information providers register and store all the resource information of their local resources such as static and dynamic attributes and their specific virtual organization policies in the aggregate directories in the higher level using GIIS. Consequently, resource providers periodically update their local resource status in their registered directories. If an information provider after a specific time fails to update the directory, GIIS assumes that the information provider is not available any more and it removes the provider from the index list of validated GRIS.
MDS-2 [43, 47]	Hierarchical-Grid (tree-style)	Propose refinements of MDS-2. They follow a tree-style hierarchy within a Virtual Organization (VO) to dictate node resource information dissemination. A virtual organization is a group of Grid peers that agree on some common resource sharing policies. Every VO designates a node that hosts all the local resource information. These representative nodes register themselves in the related directories where they are organized in the hierarchy, based on specific resource information. In MDS sometimes it might happen that a low level node begins to stop updating and sending resource information to the GIIS. Thus, the MDS search (in both centralized and hierarchical manner) could not be deterministic which means if the query answer is available in the system, its not guaranteed to see the answer in the query result. In other words, the system could not provide a reliable resource information to the clients. Furthermore, as the size of the VO and the query rate increase, MDS has scalability limits due to the high network traffic near the root node.
MDS-3/4 [48-50]	Hierarchical-Grid	Proposes a distributed monitoring system for high-performance computing systems such as clusters and Grids. The resources are organized in a set of federated clusters. In each cluster, a representative node reports the status of resources within the cluster to the federated monitoring system. It leverages various technologies such as XML for resource description, External Data Representation (XDR) for data packing and transferring resource information, and Round-Robin Database tool (RRDtool) for data structuring and storing information. It has tried to achieve a scalable solution by reducing network overhead per node and increase the level of concurrency, however, like MDS-3, it still suffers from similar problems to the centralized systems such as single point of failure and limited scalability.
Ganglia [51]	Hierarchical-Grid	Proposes a distributed monitoring system for high-performance computing systems such as clusters and Grids. The resources are organized in a set of federated clusters. In each cluster, a representative node reports the status of resources within the cluster to the federated monitoring system. It leverages various technologies such as XML for resource description, External Data Representation (XDR) for data packing and transferring resource information, and Round-Robin Database tool (RRDtool) for data structuring and storing information. It has tried to achieve a scalable solution by reducing network overhead per node and increase the level of concurrency, however, like MDS-3, it still suffers from similar problems to the centralized systems such as single point of failure and limited scalability.
Resource Category Tree (RCaT) [52]	Hybrid-Grid (decentralized-tree)	According to the priorities of the resources characteristics, RCaT organizes and categorizes the resources in form of Georgy Adelson-Velsky and Evgenii Landis (AVL)'s trees (i.e., self-balancing binary search trees) where each node in the tree is responsible for managing the resources for the values of a specific attribute within a range, instead of a single attribute value. The tree is organized based on a self-balanced structure from top to down where each level categorizes the values ranges for a particular attribute. A Primary Attribute (PA) for each tree root is assigned which is the most important attribute that can define the resource's capability. This is also relevant to the application resource requirements. For example, in the case of an intensive computing parallel application, the number of cores in the processor (static attribute) or the current load of the processor (dynamic attribute) might be a potential PA. Each node in the tree only stores information about its connection links to the child nodes and parent node and also the range values which is responsible for. Therefore, in RCaT, it is not necessary to store large amounts of information in the higher nodes, achieving better scalability. However, the maintenance cost to create different dynamic trees based on different attribute priorities is high, and it has a significant impact on the overall system efficiency. In comparison to the traditional trees and hierarchical systems, the distribution of the query loads in the RCaT's trees efficiently has been balanced and also the Average Search Length (ASL) or the number of involved nodes to process a query has been decreased. In overall, we can say that RCaT has improved and enhanced the efficiency to traverse queries in tree style structure, but it remains unsolved the number of critical issues related to scalability, heterogeneity, dynamicity and even complex querying.

from a single point failure and it could be more scalable in comparison to other counterparts. The general search mechanism for a query is to explore the local Grid peer in advance. The unsuccessful query, in the next step, will be forwarded to the closest remote Grid peer in vicinity using various strategies and techniques (such as random walk or any casting).

The P2P system contains the autonomous components that are able to accomplish actions automatically and without any manual control. In decentralized Grid discovery, information requester and information provider (or resource broker) are two discovery agents which interact to each other in a self-organizing and self-adapting manner. The broker agent contains a resource information database that must be discovered from other Grid peers, in order to provide a global knowledge view to the local information requesters. P2P systems are the appropriate distributed paradigm to deal with scalability problems. Besides, they provide a significant efficiency in the context of dynamicity of resources.

Importing and leveraging the alternative concepts like P2P can improve the performance of the resource discovery in Grid [54, 55]. P2P-based approach is another common model for resource discovery which offers a significant advantage over their hierarchical counterparts by the way of resistance to failure and traffic congestion. In particular, structured P2P systems based on Distributed Hash Tables (DHTs) are very popular for file-sharing applications, but not for sharing resource information. Moreover, typical structured P2P-based systems are very sensitive to churning leading to resource unavailability [56]. These systems achieve good performances and scalability characteristics, but they are limited to only support exact matching. Moreover their hashing functionalities performs well with static attributes, but they need to be enhanced for handling dynamic objects appropriately. SWORD [57], is a P2P-based approach which supports multi-dimensional resource attributes and range-based querying, and improve the existing systems by resorting to multiple DHTs.

The combination of P2P and Grid RD models [25, 58–60] would be desirable to build fault tolerant and large scale distributed systems. There are two kinds of approaches in this field which are based on structured and unstructured overlays networks. The first model uses a unstructured overlay network with flooding based query propagation. One of the advantages of this approach is the ability to perform resource discovery with high expressiveness. However, the discovery systems are not exhaustive and efficient. The response time of the queries is high due to flooding and blind search. Also, rarer resource information may be unable to be found. NodeWiz is a sample of this group. [61] is a review of the most promising Grid systems that employ P2P strategies to facilitate resource discovery.

D) Hybrid-Grid: Hybrid approaches have been specially designed to provide a high level of scalability and fault tolerance, which is required in large scale environments. They are proposed to enhance and extend the natural capabilities of the centralized and hierarchical models such as flexible querying, reliability in terms of deterministic search and discovery correctness to a wider set of desired features and capabilities in different aspects by aggregating to the advantages of the fully decentralized models such as scalability, reliability in terms of single point of failure, self-organization and efficiency. Decentralized-tree and decentralized-mesh are the sample architectures which have been used to deploy hybrid solutions like RCaT [52] for resource discovery in Grid.

2.3.2.2 Peer to Peer Systems

In the previous section we discussed the usage of P2P strategies for managing resource discovery in Grid Information Systems. Here, we will address the more general aspect of resource discovery in P2P systems. Peer to Peer computing systems have emerged as an alternative paradigm to construct large scale distributed systems. The concept of P2P introduces a number of significant advantages in different aspects of resource discovery including scalability (due to collaborative resource sharing between peers), reliability (e.g., fault-tolerance)

(due to the commensurable essence of peers), robustness (due to self-organization against peer or system failures). For resource discovery in P2P systems, it is more common for peers to create network overlays on the top of physical network topologies. These overlays might provide a specific structure to store and distribute resource information among peers, or the peers can follow dynamical information maintenance and distribution behaviors in an unstructured ad-hoc fashion. Table 2.2 elaborates some recent examples of P2P resource discovery approaches which are based on different overlay architectures. In continuation of this section, we discuss the different approaches in P2P, which we organize in structured, unstructured and hybrid.

Table 2.2: Examples of resource discovery in P2P-based systems (DS: Discovery System, FoCs: Flocks of Condors, MaT: MatchTree, CyG: CycloidGrid, SkipC: SkipCluster).

DS	Base	Mechanism
MaT [62]	Hybrid Overlay	P2P Proposes a scalable and fault tolerant system by creating a self-organized tree for query distribution and result aggregation with a specific asymptotic latency increase pattern. It reduces the query latency and improves the system fault tolerance through redundant query topologies, sub-region queries, and dynamic timeout policies and set of dynamic timeout policies. It supports complex queries and guarantees query completeness.
Tripod [63]	Hybrid Overlay	P2P Proposes an inherent fault-tolerant system based on hybrid overlay network in order to enhance the scalability and search efficiency of resources in highly dynamic large scale heterogeneous environment. It is efficient to discover resources in proximity while it provides dynamicity in terms of dynamic resource attributes. Additionally the network overhead cost for overly construction and maintenance is very low.
CyG [64]	P2P overlay	Provides a two stages Quality of Service (QoS) and locality aware discovery algorithm for P2P based volunteer computing systems. In the first stage, it discovers a set of resources based on the required quality of service and the current load of the peers, and in the next stage it selects the closest resource in terms of communication delay (latency) between peers which is calculated using a network model based on queuing theory with considering the background traffic of Internet.
PIRD [65]	DHT-based	Focuses on multi-attribute querying and locality awareness. The system has been built based on a hierarchical P2P structure which uses a locality sensitive hashing to generate indices for the peers and resources in a DHT overlay. It leads the system to be able to discover resources geographically near to requesters. PIRD incorporates the Welch algorithm [66] to compress attribute information and then it weaves all attributes into a set of indices using the hash function. Thus the system is able to efficiently define and perform a multi attribute query by sending a single query message to the system.
FoCs [67]	Pastry (DHT-Based)	Combines the flocking of the condor pools with the Pastry mechanisms. It uses a self-organized Pastry overlay to index the distributed condor pools. The system is static and doesn't support dynamicity
SkipC [68]	Hierarchical P2P Overlay, Pastry (DHT-based), SkipNet [69], SkipGraph [70]	Designed to solve the problem of range querying and other kind of proximity-aware related complex querying in classical DHTs. The system is able to support both exact matching and multidimensional range querying without storing extra information in the peers. It has two tiers hierarchical architecture where in both of them the peers are ordered and organized based on the sequence of their peer identifies, considering the point that the semantically related peers are stored near each other without hashing their resource keys. The pointers to the super peers in the higher tier are stored in a new data structure called Triple Linked List (TLL) which are used to find the longest prefix for the query key in the remote clusters. In the lower tier, the routing table of each peer contains pointers with exponentially incremental distance.

Structured Systems: Most P2P resource discovery systems depend on a structured architecture (e.g., ring, tree) where the system can be understood by certain nodes of which the resources information are fixed. Such systems are in general faster in discovering resources than unstructured RDs, with a fixed and predictable time of resource discovery. These approaches can be extremely effective for searching resources using unique identification, but they fail when a search using partial matching is required. Moreover, they create additional overhead due to management of the network architecture (by updating the system structure in the case of node failure or arrival).

There are two types of structured overlays: DHT based systems such as Chord [71], Content

Addressable Network (CAN) [72], Pastry [73], Tapestry [74], P-Grid [75] and D-Grid [76] and non-DHT based solutions like Mercury [77]. However, most overlays of this type commonly make use of DHTs. A DHT is a distributed data structure to efficiently perform distributed storage and lookup of pairs (key,data) which provides a scalable, fault tolerant and fast locating of data when a key is given. Here we explain some of the well known DHT based solutions in detail.

Chord organizes peers and resources in an m -bit, ring based, identifier space where m represents the total number of required bits to specify each fixed-length keys/identifiers and the maximum number of 2^m keys (representing resources or entities) can be identified in the interval of $[0, 2^m - 1]$ (a range of non-negative integers). We also must note that the value of m should be large enough to avoid collision in hashing. The Chord ring is the basis for efficiently locate the peer that contains a particular data item (i.e., resource description or resource information). Both peers and resources are assigned m -bit identifiers and ordered based on their identifiers in a circle modulo 2^m (called Chord ring) by employing SHA1 (Secure Hash Standard), a consistent hashing function, which hashes the peer's location (IP address) and the key (e.g, definition of a resource by a keyword, summary of the resource information) respectively. Each key is assigned to the first peer clockwise from the key in the Chord ring (called successor peer of the key) whose identifier is equal to or pursues the identifier of the key. Predecessor also is the peer that takes position right before the key in the ring. The distribution of the keys between peers are load balanced by roughly allocating all the peers an equal number of keys where each peers must maintain the correspondent data (resource description) for several resource keys in the form of (key, data) pairs.

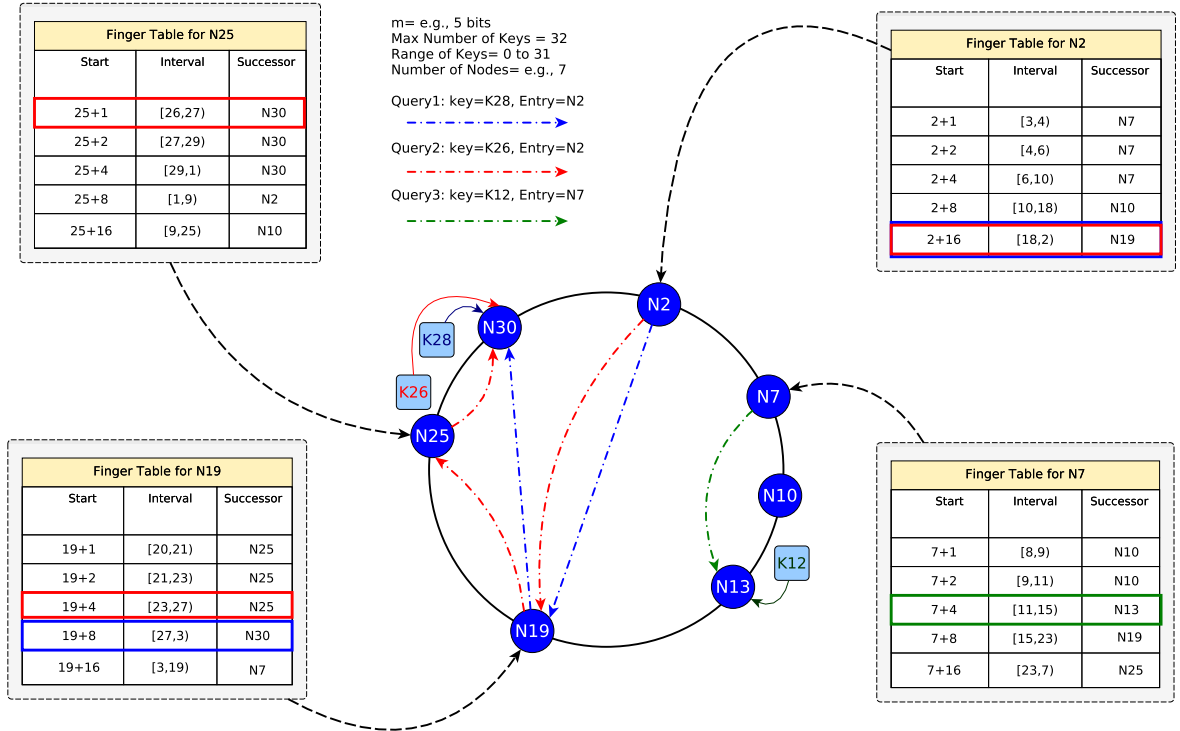


Figure 2.7: Examples of query processing in Chord.

The process of locating resources can be implemented on top of Chord overlay by allocating a key with each data (resource description) item, and storing the key/data pairs at the correspondent peers. It is started by issuing a query to the system, afterwards, the query hashed to a specific key identifier by consistent hashing function, given a key, considering the point that each peer knows about its successor, the queries can be forwarded around the ring using the successor pointers until the successor peer who is the owner of the key is encountered. The successor peer of the key is responsible to store the required resource information related to that key in the query. In order to enhance the efficiency of the algorithm and limit the number of explored peers to handle a query, each peer maintains an m -entries small finger table (see Figure 2.7), which contains the pointers to the potential successors for each range of keys.

A new peer can join the system by contacting a known peer already included in the Chord ring. The joining request can be responded by the address of the potential successor and predecessor, afterwards, the new peer will ask these peers to be added in the ring. Consequently, the number of certain keys which were previously assigned to the successor peer will be moved to the new peer. Similarly, when a peer voluntarily departs the system, it transfers its keys to its successor peers in order to maintain the resource availability of the system. Sometimes, it might happen that a successor of a peer suddenly leaves the system, in such a case each peer can substitute the ring with another successor in the list of alternative successors which already have been stored in each peer.

Sudden involuntary leave of peers can affect the resource availability due to the fact that resources in the Chord system are not replicated crosswise peers. Thus, the system can designate an alternative peer as the replacement for the departed peer. But there is no guarantee to find an equivalent resource in the system instead of the resource which already becomes unavailable due to its peer's departure. This will decrease the reliability of the Chord system to a certain amount. Moreover, to address the problem of frequent departures and joins of the peers in the system, Chord uses a stabilization protocol which runs on each peer periodically to maintain the successors information and finger tables updated. However, this is a costly process which affects the system efficiency by occupying the bandwidth to handle the maintenance loads. Chord distributes the resources among the peers in an appropriate load balanced fashion while it reduces the bandwidth cost of the query by avoiding flooding. The combination of these performance factors result that the system becomes extremely scalable, additionally exploiting the capability of the finger tables leads the Chord to facilitate faster and efficient querying.

However, the Chord DHT has number of disadvantageous, for example, due to using the hash values, it is only able to perform exact querying, while it can not resolve a reliable complex query (e.g, range, multidimensional and keyword search). Furthermore, the peers in Chord based resource discovery solutions (like pSearch [78]) do not have full autonomy to control their local resources, thus the Chord system could not be a purely decentralized system. Research works in [79–86] discuss some examples of improvements on Chord. Figure 2.7 demonstrates examples of query processing using Chord.

Pastry [73] and Tapestry [87] are two distributed lookup systems which are very similar to Chord. The difference is that in Chord, query forwarding is performed based on the numerical differences between the source and destination address while in Pastry and Tapestry the request forwarding is based on prefix and suffix based routing approaches respectively. Moreover, despite similarity to the Chord, they provide support for locality-aware discovery, which ensures that peers and distances in the logical overlay network have a correlation with

the physical nodes and network distances in the underlying layer. Therefore, passing a query along a logical connection to a close peer in overly does not lead to a long distance query traveling in the underlying network.

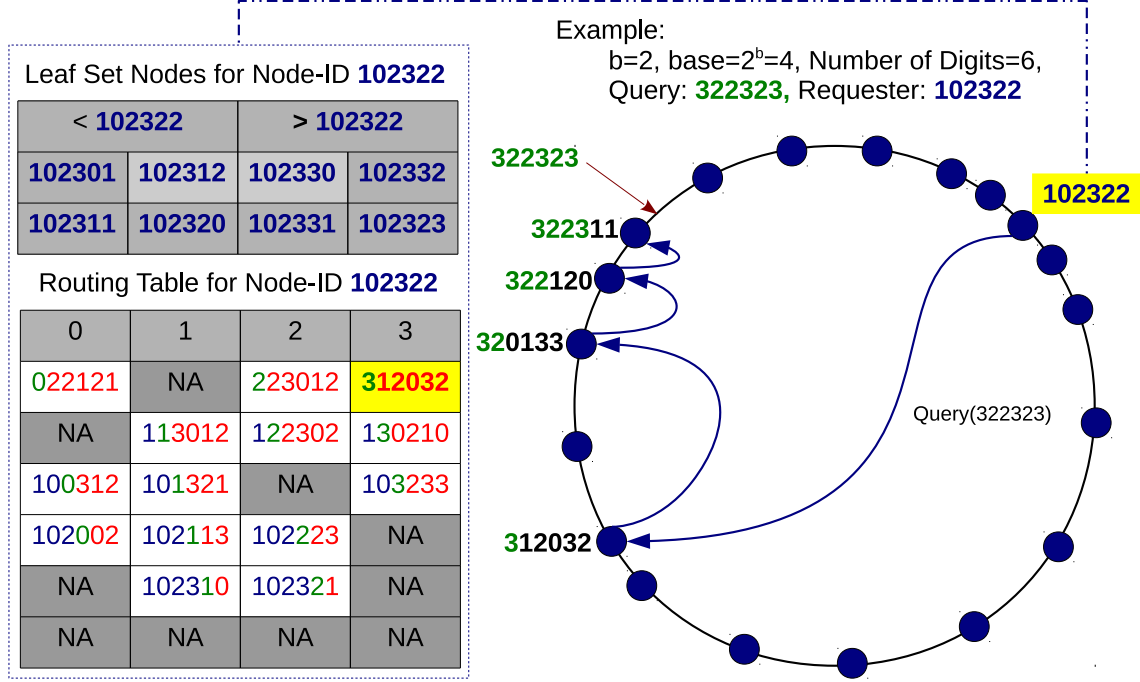


Figure 2.8: Routing example in Pastry: Query with the key 322323 arrives to the node 322311 which has the closest node-id. The routing algorithm corrects one digit at each step and then uses "leaf-set" to locate node with closest node-id to target.

Pastry creates an m -bit (generally $m=128$) identifier space in the range of $[0, 2^m - 1]$ where using a cryptographic hash function, the IP address or the public key of each peer is hashed to a unique node-id. Consequently, the peers and resources must have their assigned m -bit node-ids and key identifiers respectively and like other DHTs the keys should be distributed among the peers in a load balanced manner. The peers are organized and arranged in ascending order of node-ids in a ring. Given a key for lookup, the system routes the query to the peer whose node-id numerically is closest to the m -bit hash value of the key. In order to implement the lookup procedure and query forwarding on the top of Pastry ring, each Pastry node is required to store three type of data items, routing table, neighboring table and leaf set nodes. The routing table of each peer stores the pointers to the (usually long distance) peers in other prefix realms which share different lengths of prefix (first i digits) with the node-id of the peer within the system.

In Pastry, ring consists of N nodes with definition of b bits length (for each digit) and the identifier $base=2^b$, the routing table of each node has $2^b - 1$ columns and $\log_{2^b}(N)$ rows. The entries at row i point to peers that share the first i digits of the prefix with the node-id while the maximum entries for each row is $base - 1$ and the $i + 1$ th digit of the entry in cell (i, j) must be equal to the column number of j . In other words the entry of each cell (i, j) of the routing table must contains the topologically closest node with prefix length i and

$digit(i + 1)=j$. The leaf set nodes contains the pointers to the l nodes which their node-ids are numerically the closest nodes ($l/2$ nodes with a numerically closest larger node-ids and $l/2$ with closest smaller node-ids). The neighboring table stores the node-ids of the peers which are topologically nearest (e.g., in terms of latency or network hops) to the current node. The neighboring information generally is achieved by caching of nearby candidates for routing table during the construction of the routing table. The pastry algorithm to find a key k in the system starts by submitting a query to node p which already included in the ring. Node p checks its local leaf set nodes to find if a match for the key k is within the leaf set, if so, the query is resolved by forwarding the request to the matched node which already owns the required resource of the query k . Otherwise, referring the information within the routing table, Pastry forwards the query messages to a node which has one more matching digit in the common prefix (see Figure 2.8).

In the rare case when node p is not able to determine another node in the routing table which provides a longer length of matching prefix, the query is forwarded to any node that is closer to the key than the current node-id within the merged set of routing table, neighborhood set and the leaf set. Research works in [88–93] are some examples of the improvement works based on Pastry.

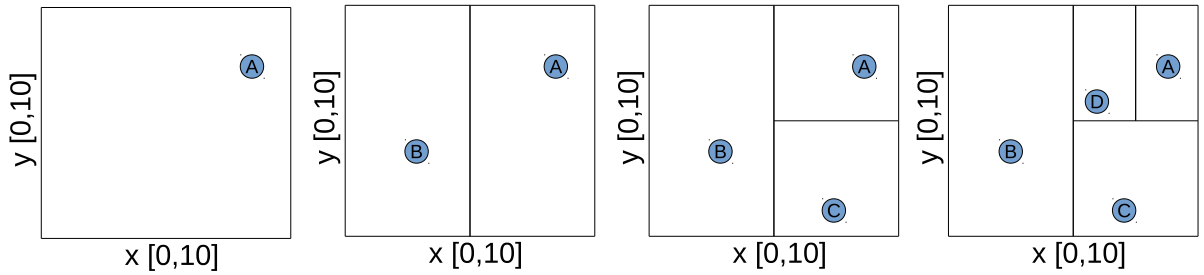


Figure 2.9: Example of partitioning in 2-Dimensional CAN. Each node is assigned a unique zone in the M -Dimensional coordinate space. A new node is usually joined by identifying and splitting a zone in the coordinate space that can be split.

CAN is another DHT which maintains a routing table including the IP address and virtual coordinate zone of each of its neighbours. It divides the search space to multiple partitions and accordingly routes a message towards a destination point in the coordinate space (see Figs 2.9 and 2.10).

The scalable P2P models such as Chord, Pastry, Tapestry and CAN have two general disadvantages, first, they do not provide fully local control over data in each peer which reduce the overall decentralization, autonomy and flexibility of the system. Secondly, they do not guarantee that the routing paths will remain permanently within a constant administrative domain, which reduces the reliability and increases the maintenance cost of the system. There are some research works in P2P such as SkipNet, SkipGraph, P-Ring [94, 95] and Online Balancing [96] which concentrate to solve above issues.

Overall, we can conclude that DHT based systems build efficient query execution structures that are well suited to deal with the scalability and efficiency issues for discovering resources in LDCE. However, they come with a number of shortcomings in different aspects such as semantic querying, dynamicity, heterogeneity and topology mismatching. There are several research works focusing on improving the capabilities of the DHTs along the line of the aforementioned

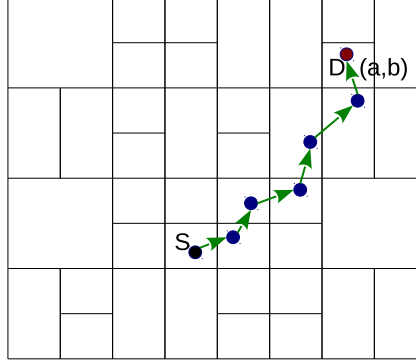


Figure 2.10: Sample routing path from S to D in a 2-Dimensional CAN. Using the greedy routing strategy, the query message, in each intermediate node, is routed to a neighboring node that is positioned closer to the desired location.

problems. In the following, we explain the drawbacks and some of the potential solutions found in the literature.

DHTs generally do not support semantic querying and keyword search. The lack of semantically-rich resource descriptions and capability to perform flexible/complex querying in DHT limited the discovery efficiency, specially in the environment where the resource requesters are not able to precisely clarify their resource requirements and conditions. For example, considering that a requester is interested in locating a certain number of vector processors (e.g., SIMD type processing cores) in an attribute-based DHT system where the resources are described based on a constant number of attribute values, the query might not be successful even if the qualified resources are present in the system. The reason is that simple exact query results require the query itself to be described precisely by the resource requester, and the resource requester does not want to limit its potential resource options, while the requester doesn't know about the available resources in the network. Semantic-based querying is one way to overcome this problem by organizing peers/clusters based on their contents (i.e., based on the conceptual similarity of the resources) in order to enhance content search.

ERGOT [97] presents an approach to overcome the problem of semantic querying in DHTs by leveraging the DHT mechanisms to build a Semantic Overlay Network (SON) [98]. DHTs are scalable and fault tolerant and they guarantee efficient lookup at a exactly predictable cost while they are semantic-free, while SONs are flexible to perform semantic driven query and are able to go beyond exact matching (but are less scalable than DHTs). It has to be taken into account that the performance of SONs highly depends on the way that semantic links are created, and the communication cost to create the semantic links. ERGOT enables semantic-based resource discovery in distributed infrastructures such as Clouds and Grids. It employs sorting of the semantic annotations (i.e., it is required to define the notion of similarity) in order to enhance and enrich the description of resources/services and queries by two strategies:

- First, peers can construct the semantic links during normal interaction of peers in DHT by recognizing the peers with similar content, thus the semantic links (links between peers with similar content in SON) can be created without any extra cost. In other words, resource providers advertise their resources in the DHT according to their annotations which results in

building SON among the resource providers which provides similar resources.

- Second, the system is able to use the advantage of DHTs to perform exact matching while it is able to resolve semantic queries based on resource match-making by measuring the similarity, between query's resource requests and resource descriptions.

Similarly, DHT Information Service (DIS) [99] proposes a scalable DHT and ontology based information service for Grids in order to speed up querying and enhance query precision and integrity, by aggregating DHTs and semantic-based query techniques. DIS is concerned in the dynamic attitudes of the resources (e.g., frequent joining and leaving of resources) in virtual organizations since it requires powerful self-organization capability of the system to maintain the system structure. Additionally, in the case of creating key space for the DHTs, a large scale key space might produce an unbalanced workload distribution among the nodes in the DHT, while a small key space reduces overall system efficiency. Thus, it is desired and complicated to arrange a moderated key space for the DHT ring where the size of identifier space is optimum. DIS builds a moderated DHT ring considering the VO mode in terms of stability of resources. The stable VOs in the system directly participate to organize resources in the DHT ring. On the other hand, for the purpose of keeping the size of identifier space moderated, only stable VOs can join the ring through a new DHT node and the unstable VO join DIS by becoming a sub-domain of the other VOs. Using the above strategy and ontology-based query techniques, DIS has extended the DHT capabilities to support semantic querying.

Another drawback of DHTs is the lack of support for dynamic attributes. For example, the dynamic changing resource information of the idle CPU cycles is not possible to be hashed in a DHT overlay. Only the information about the static attributes can be stored in DHTs. Furthermore, DHTs do not support heterogeneity of resources in terms of the number and types of resource capabilities and functionalities, for example, processing resources provide different types of attributes than communication and memory resources, and even the types of attributes for Graphics Processing Unit (GPU) resources might be different from CPU resources. In fact, nodes types homogeneity is an implicit assumptions in most of DHTs such as Chord, CAN, Pastry and Tapestry where they expect all the nodes have the same behavior and capabilities. It results to the limited efficiency and applicability of discovering resources in LDCE saturated by heterogeneous resources.

Topology mismatching or the lack of support for proximity-aware querying is another common problem in DHTs. One of the important expected features of resource discovery, specially in computing environments, is the ability of the querying system to discover resources in close vicinity. Pure P2P based systems, like DHT-enabled approaches, generally provide limited (e.g., Pastry) or zero (e.g., Chord) support for such proximity-aware querying. The neighboring nodes in DHT overlay are not necessarily close to each other in the physical topology. The DHT overlay structure destroys data locality, which increases the discovery overhead specially to process a particular type of queries such as resource graph query and range query.

Clustering and hierarchical based solutions (e.g., super peers) have been proposed to resolve the aforementioned DHT shortcoming. Super-peer based discovery systems aimed to provide an optimum solution in order to achieve a balance between the built-in efficiency of the hierarchical/centralized system, and the load balancing, self-organization, and fault-tolerant features provided by P2P based discovery systems. Kazaa [100] is an example of the super-peer model which designates the more stable and powerful nodes as super-peers. The new member must find the closest existing super-node in the network and establish the overly connection to

that node. Kazaa originally has been used for file sharing purposes. However, its concept can be extended to use for general resource sharing, even of computing resources. Upon joining a new node to the overlay, it sends its list of resources to the super-node. The super-node registers the resources information of the individual nodes in an indexed directory. Each requester sends its requests to the super-node. Afterwards, the requester searches for the requested resources in its local index directory and, if the desired resources are not locally available, it propagates the query to another super-node in the system. Kazaa is also able to perform keyword search and provides some flexibility for querying. The works in [58, 101, 102] are other examples of super-peer based discovery systems. These solutions enable efficient discovery of the resources belonging to one super-peer, however, they do not introduce any extra improvement for P2P based resource discovery over several super-peers.

Proximity-aware clustering approaches such as TriPod [63] organize the close resources in the same cluster while other semantic-aware clustering approaches [52, 103] organize the semantically similar resources in the same cluster. Depending on the strategy, clustering can preserve both locality and similarity features.

Unstructured Systems: In unstructured discovery, the distribution of the resource information among the nodes is not followed by a predefined or controlled mechanism. An unstructured system is able to support partial matching and also complex querying. The use of a loose architecture makes the system resistant to node failure and Denial of Service (DoS) attacks. Furthermore, the convenient system adaptation to the frequent node joins and disjoins provides dynamicity for querying in such environment. However, unstructured discovery has a low rate of resource discovery in comparison to structured systems. Nodes in these systems are free to behave as they want and they carry a part of the network functionality, thus, their failure or misbehavior can be costly. Besides, an unstructured approach has numerous challenges specially in terms of fault tolerance under churn, load balancing and flash crowds. Unstructured discovery in P2P possess the following distinct properties:

1. Decentralization: implies a system architecture without any centralized server or control point. It avoids single or multi points of failures in P2P environment.
2. Self-organization: contains the capability of the system to be dynamically reconfigurable, which facilitates forming a dynamic network overlay while the system keeps changing over the time.
3. Autonomy: the peers in the system are independent entities, which have autonomy to make decision based on their preferences and priorities, to establish or cut a connection link to other peers in the overlay network.
4. Anonymity: the peers in the system do not have a global knowledge about the whole system. In other words, each peer only knows about itself and no one knows about the others (even neighbors).
5. Unstructured: there is no predefined structure or fixed points to store and access the resource information.

Accordingly, depending on the search methods (see Section 2.4.3), unstructured discovery solutions (e.g., Gnutella [104], JXTA [105], Freenet [104], Morpheus [106, 107] and Routing Indices (RI) [108]) can be categorized in two groups: deterministic and none-deterministic discovery.

Gnutella [109, 110] is a purely decentralized resource discovery based on a pervasive exchange of messages (aggressive flooding) which originally has been designed for file sharing. However, it has been extended to cover for the domain of resource sharing in the computing environments. Gnutella consists of a set of Gnutella Nodes (GN) which are called servants. They play both roles of client and server entities. Each servant is responsible for processing the received queries, check for matches against their local set of resource information and respond with related results. The search mechanism relies on broadcasting and back-propagation communication. It starts by sending a “Query” message contains a criteria string and a randomly generated message identifier, flagged with NHP (number of passed hops) and Time To Live (TTL) fields from a requester to all of servants in its vicinity. Upon receiving a “Query” message in the target node, it decreases the TTL value in the header descriptor. If the query conditions are not satisfied with the current resources and the TTL after decrement still is not zero, the servant broadcast the query message along the (open TCP) connection links to its neighboring servants, except the link that query came from. Additionally, each GN uses a caching mechanism to maintain the track of the recent queries for the purpose of back-propagation of the result messages (or “Hit” messages) to the original requester, and also avoid to duplicate forwarding of the same queries.

GNs must always keep information about their neighbors updated. For this purpose each node periodically broadcasts “Ping” messages to its neighbors and the receivers answer with “Pong” messages to confirm their presence. GN also supports “Push” message which is a request that can be send to the resource providers in order to ask them to contact the resource requester. Gnutella, provides efficient properties specially in dealing with dynamicity and heterogeneity of resources. However, it has significant number of drawbacks: the search is none-deterministic, the TTL technique implies a better scalability but reduces the number of explored hosts for a query. Specially for the rare resources, the limited query radius leads to the limited number of hits (successful discovery). Increasing the TTL on the spot a query is not resolved is a technique to solve this problem, but it results in a logarithmically increase of the network traffic, which could not be a scalable approach in dense communication systems. The other shortcoming is that Gnutella does not support exact matching. The search is based on keyword querying, which is not accurate. The protocol also is not efficient in terms of resource consumption due to its flooding nature (broadcasting) as it invokes too much nodes to handle a query. There are a number of research works [111, 112] that proposed alternative methods and techniques to improve the Gnutella drawbacks.

Routing Indices (RI) [108] uses distributed indices in unstructured P2P networks. The advantage of this mechanism relies in the fact that queries are disseminated and forwarded only among the places of the network where resources existed, thus avoiding to flood query requests to the nodes which are not useful. The main drawback of this solution is that this indexing system comes from the presence of cycles in the network graph. A recent work [113] of this type extends RI and proposes a technique to perform resource discovery in grids based on P2P with capability to perform multi-attribute queries and range queries for numerical attributes. It uses an information summarization technique presented in [114] and creates different types of summaries and accordingly presents a metric (called goodness function) needed by RIs to guide the query process. It still suffers from RI drawbacks as well as lack of support for complex querying. A similar proposal [115] presents a task/job-level resource discovery with limited flexibility of querying to handle multi-core machines in desktop grids. This technique handles resource availability based on a few set of numerical parameters, such as CPU speed, number of cores, and memory availability. We must note that most of the

proposed unstructured discovery solutions in the literature are initially designed for file sharing applications which is out of the scope of our interest in this thesis.

Hybrid solutions try to combine and strengthen the benefits offered by the classical structured and unstructured resource discovery systems. MatchTree [62] and Tripod [63] are examples of hybrid solutions (see Table 2.2).

Table 2.3: Summary of comparison of the major types of resource discovery for different performance factors (PFs: Performance Factors, RB: Reliability, FT: Fault-Tolerance, LB: Load-Balance, AN: Autonomy, CQ: Complex Querying).

PF	Centralized-Grid	Hierarchical-Grids	Structured-P2P	Unstructured-P2P
Scalability	Not scalable specially for the large scale Grid because of the single point of failure bottlenecks	Provides better scalability than centralized system, but still suffers from overloaded hot spots; the roots of hierarchy becomes easily single point of failure	Limitations due to the large amount of overhead network traffic	Generally scalable
Efficiency	Provide efficiency in terms of fast discovery and precise discovery results	Similar to centralized	Fast lookup speed. Guarantees to perform a query in a bounded number of hops. It assumes guaranteed the availability of the resources in the network, thus it doesn't provide a good efficiency to work in dynamically changing environment. The dynamic changes of resource information must be propagated over the network which reduces system efficiency by creating a considerable amount of network overhead.	Low lookup speed. Costly membership management. Slow propagation of information in the case of dynamic values. Does not assume any guarantees about the peers or resource availability, therefore it does not put any constraints on the topological placement of resource information.
RB	Highly reliable in terms of result accuracy. Not reliable in terms of single point of failures	Reliable in terms of result accuracy. Not reliable in terms of single point of failures.	Reliable in terms of exact matching. Not reliable in terms of interval search	There is no guarantee to perform successful querying. Reliable in terms of dynamically changing environments
Dynamicity	Supported in terms of dynamic attribute. Not supported in terms of indexing mechanisms and periodical updating of the resource information	Supported in terms of dynamic attribute. Better support of indexing mechanisms and periodical updating.	Supported in terms of dynamic topology changes. Not supported in terms of dynamic changes of attribute values where the system has problem to store rapidly changing resource information. The overlay networks are appropriate to maintain static resource information while for dynamic information the overlay must be reorganized which is costly.	Highly supported
FT	Not supported	Not supported	Supported	Supported
LB	Not supported	Not supported	Supported	Supported
AN	Not supported	Not supported	Supported	Highly supported
CQ	Generally provide all kind of flexible querying due to the advantage of using central data structure/-data base	Limited support	Do not support or have a limited support for interval search (such as partial querying and range querying), semantic search, multidimensional querying and resource graph discovery	Better support for complex querying in comparison to structured P2P and less flexibility than centralized systems

Table 2.3 provides comparison of major types of resource discovery for both Grid and P2P environments in most aspects. We further discuss the details of the performance factors used for this comparison in Section 2.5.

2.3.2.3 Cluster, HTC and HPC

Computing systems such as HPC, Cluster and Cloud are generally based on a centralized/hierarchical architecture, leading to the use of centralized resource discovery to respond to user requests. For example, when a request for resources arrives at a HPC cluster-head or a Cloud service provider in the front-end, the resources required can be discovered (allocated or provisioned) by searching in a specified pool of resources or in a back-end data-center where the number and type of resources are known beforehand (this may not be necessarily true in Clouds). For such environments, resource discovery based on a centralized architecture could become an easy task. And, in fact, instead of discovery problem, the resource scheduling issues are dominant. In this and the next section, we aim to discuss resource discovery for HPC and Cloud. However, due to the nature of these types of environments, we may also discuss relevant topics such as resource scheduling.

A Computing Cluster is defined as the composition of several computing nodes (workstations), multiple storage devices and a number of interconnections, which together appear to the users as a single system that is highly available and reliable to run user tasks. HPC, High Throughput Computing (HTC) and Many Task Computing (MTC) are some types of computing clusters.

HTC environments provide a large amount of processing capabilities to run user jobs (i.e., independent loosely-coupled tasks), over long periods of time, through dynamic exploitation of all the existing available computing resources in the system. In fact, HTC tasks are sequential and independent. They can be individually scheduled on wide range of heterogeneous computing resources, geographically distributed across multiple administrative domains. Various Grid Computing techniques can be used to build HTC systems [116]. HPC creates supercomputers to run tightly coupled parallel tasks on large amounts of computing capacity over short periods of time. HPC focuses on fast execution of tasks (generally dependent tasks), therefore the interconnects of HPC clusters must provide low latency for communication between processes running on different computing resources. MTC is the bridge between HTC and HPC, with emphasis in employing many computing resources over short periods of time, in order to execute many computational tasks that could be either dependent or independent.

Through using traditional cluster management systems, it might be possible that each cluster nodes knows about the static attributes of the resources in other nodes without relying on a complete approach for resource discovery (for example, ignoring the dynamicity issues, a centralized cluster can statically be configured to maintain all the information about the cluster resources in a central server and the clients just simply submit the queries to the server). In some cases the static information is not really important for the task allocation in the cluster. In other cases, it would not be enough since the most important and critical requirement in these cases is the dynamic information, to determine whether the required resources are currently free or are occupied. However, state information (i.e., the information about the dynamic resource attributes) such as processor availability, memory usage or available bandwidth, which are rapidly and frequently changing, may not be accessible without leveraging an efficient resource discovery mechanism.

Resources may have multiple single attributes which can be either dynamic or static. Thus,

depending on the computing environment and the purpose of resource discovery, different discovery strategies can be implemented, ranging from single-attribute to multi-dimensional discovery systems. CPU availability (CPU load or CPU utilization) and memory utilization (memory usage) are two important dynamic attributes which have the key roles to structure any computational resource discovery mechanisms. In fact, task allocation is not feasible if the qualified discovered resources would be unavailable in terms of CPU and memory load. In general, the resource discovery problem in computing clusters aims to find the available computing resources in the system for job scheduling and task allocations. CPU discovery and memory discovery are the examples of single-attribute discovery (i.e., state discovery) where CPU utilization and memory usage respectively is the only resource information of interest.

In cluster computing, we can categorize resource discovery approaches (in the case of state information) in three groups: Active, Passive and Predictive. In the Active approach (e.g., centralized batch systems and decentralized state discovery algorithms), the requesters use a intrusive method (within a client-server like communication model) to know about the state of the other resources. Additionally, the Active approach generally does not support dynamic load balancing. In the Predictive approach, the requesters use the state of the resources in the previous cycle (which has already been obtained using an Active method) to predict the state of the resources in the upcoming cycle and thus reduces the overhead of the Active method. The Active approach generates a large network overhead due to the periodical resource state updates with direct communication between the resource manager and the compute nodes. The Passive approach attempts to get the information about the state of the resources by analyzing the behavior of the existing network traffic in the cluster. Since the Passive approach is not a intrusive mechanism, it doesn't generate extra overhead, which results in a decrease of the communication complexity (the number of messages required to solve a discovery problem), but provides less accurate information.

In cluster computing, another alternative for Active discovery is using decentralized algorithms such as ALG-Flooding [117], Swamping [117], Random Pointer Jump [117] and Name-Dropper [117], that are applicable to gather the state information of the resources on different nodes around the network. The state information denotes that if a particular resource is available (in terms of CPU or memory) to allocate an specific task. We discuss these algorithms in more details in Section 2.4.3.

The Batch System (or Asymmetric Computational Clusters) [118–120] is a conventional type of cluster management systems which has been used to compose most of the current large scale computational clusters. It generally contains three types of basic components: Resource Manager (containing resource capabilities and status), Queue/Job Manager (containing creation, queuing, controlling and monitoring of the user's jobs) and Scheduler (containing resource mapping and allocation) (see Figure 2.11).

The queue manager lets users submit their jobs to the queues, and provides possibilities for users to monitor and control the state of their jobs, which can be running on the computing nodes or waiting in the queues. For each job, the scheduler selects the target computing nodes and then assigns the jobs to the resources on the computing nodes according to the scheduling policy. In the next step, the resource manager monitors the state of the resources on the computing nodes as well as the state of the job executions in the resources. The computing nodes periodically update the resource manager regarding the state (e.g., available, occupied) of their resources or the resource manager can be responsible for getting this information through periodical querying of the computing nodes. Some examples of the batch systems are Condor [121–123], Berkeley Open Infrastructure for Network Computing (BOINC) [124],

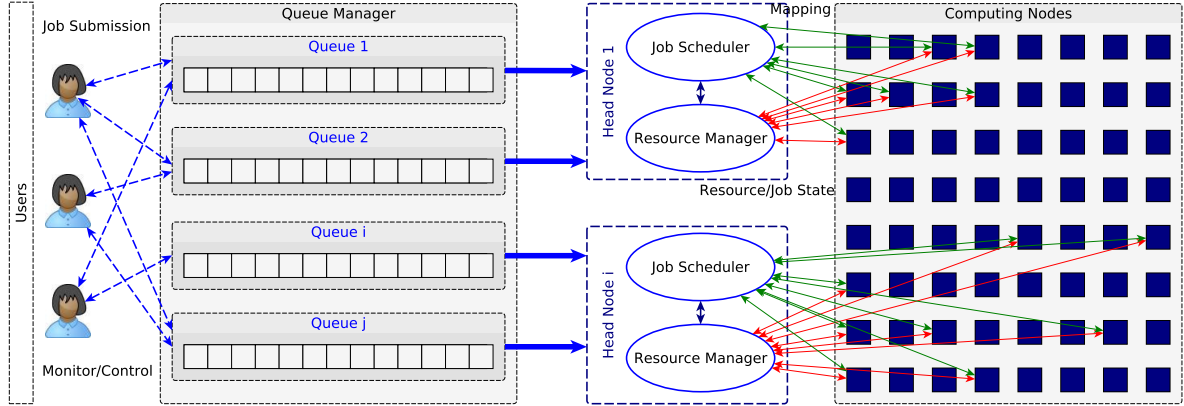


Figure 2.11: Computing strategy in Batch Systems.

Globus [125], Fast and Light-Weight Task Execution Framework (FALKON) [126], Oracle Grid Engine (previously known as Sun Grid Engine or SGE), Dodo [127], and Tera-scale Open-source Resource and Queue Management (TORQUE) [128].

Condor is a centralized batch system that is designed specially for HTC clusters. Like other batch systems, it has three main components: User Agents, Resource Agents (or Resource Owner Agents) and Matchmakers (see Figure 2.12). User requests are represented within User Agents, so the User Agents submit their resource requests to the Matchmakers which includes all the constraints that define the characteristics of the desired resources. For example, a User Agent (requester) may need to only find the resources running a specific operating system. On the other hand, resources are represented within Resource Agents, which contain the resource owners policies. For example, a resource owner may only be willing to serve the requests made by a specific group of users. The Resource Agents advertise their resources through resource offer messages to the Matchmaker where it attempts to find the matches between resource requests and resource offers in a way that all the constraints of the three parties (User Agent, Resource Agent and Matchmaker) are satisfied.

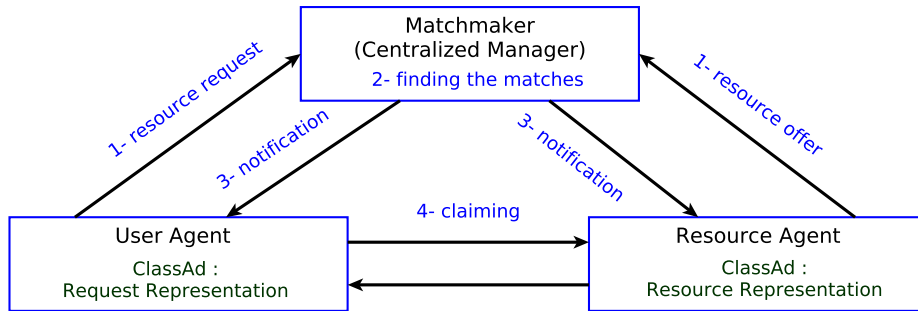


Figure 2.12: Condor matchmaking.

The Matchmaker also implements a collection of system-wide policies including logic to map the resource requests to the resource information through imposing its own constraints

in the matchmaking process. For example, the Matchmaker generates the priority ranking for the requests, thus the requests with higher priorities have greater opportunities to find the matches. When a match is found, the Matchmaker notifies both User and Resource Agents, which consequently will execute a claiming protocol to start the task allocation process. The Matchmaker is also able to measure how well a resource can satisfy a request by running a preference function. The resource information are expressed in ClassAd format, which is used by the Matchmaker to facilitate the matchmaking and resource-preference measurement. Additionally, Condor provides features to support resource scavenging: a capability to locate idle resources using a daemon providing summarized monitoring information such as average load, total idle time, as well as some other elements of basic information.

BOINC is a centralized loosely coupled HTC platform (i.e., desktop Grid like Entropia [129]) that provides resource sharing facilities for large number of volunteers to create, operate and monitor the public-resource computing projects such as **@home* (e.g., SETI@home [130], Predictor@home, Folding@home [131] and Climate@home). BOINC encourages the creation of many projects while it encourages volunteers (the computer owners around the world) to participate in one or more projects. Resource owners can enforce their own policy to specify how their resources are allocated to each project. Each BOINC project is identified by a single master URL which serves as a directory of scheduling servers, while resource owners can register their resources and participate in the projects. BOINC, similar to other batch systems like Condor, relies on knowing state information of all computational resources, which resulting in a large amount of information that limits system scalability. In fact, BOINC suffers from a scalability problem since it uses a global scheduling mechanism.

TORQUE is a Workload and Resource Management System (WRMS), which consists of two components: one for job execution and resource monitoring, and other (called Moab) for queue management and scheduling. These two components together provide all the necessary functionalities and meet the requirements of a batch system. TORQUE uses a one-to-one mapping scheme to allocate jobs to each individual compute node available, which is not efficient because each compute node can perhaps execute more jobs. TORQUE is also not able to dynamically balance jobs among resources. FALKON is derived from TORQUE aiming to enhance its efficiency and is able to assign multiple jobs to a single compute node. However, like TORQUE, it doesn't support dynamic load balancing. FALKON integrates multi-level scheduling and streamlined dispatchers to increase the system performance by decreasing the task execution time. It attempts to achieve higher scale through renunciation of some functionalities such as priorities and multiple queues (which already have been provided by Condor). However, since the FALKON dispatcher is centralized, its scalability is limited.

Dodo [132] is a batch system for harvesting idle resources (in terms of memory) in off-the-shelf clusters of workstations which includes the components such as, central manager, scheduler, resource monitor and the idle memory daemons. In this system, resources periodically inform the central manager regarding their state (free or busy). The system only schedules the jobs to the resources on the idle compute nodes, therefore it can not provide support for dynamic load balancing.

2.3.2.4 Cloud Computing Environments

Cloud computing [12, 133, 134] provides infrastructure (Infrastructure as a Service or IaaS), platform (Platform as a Service or PaaS) and software (Software as a Service or SaaS) as services to the end users. It helps the users to create and manage their customized

hardware and software, while it reduces the cost and the complexity of maintaining the required hardware/software for different user's applications. Grid and Cloud computing are concepts, as discussed. Grid and Cloud computing attempt to highly utilize and manage distributed resources to efficiently attend large (computational) jobs. But, Grids and Clouds are fundamentally different in several aspects. Clouds have built-in administrative boundaries [12, 135] which affects its capabilities to be inter-operable, while Grid do not reflect such limitation in its operation [136]. Grids nodes have autonomy to self-manage their characteristics and behavior.

Cloud computing uses negotiated Service Level Agreements (SLA) to dynamically scale the user instances in the hardware infrastructure, software platform and software application according to the potential resources in the Cloud. However, the capability of a single Cloud is limited, and it might happen that a single Cloud is not be able to uniformly maintain the quality of services for all users requests. One of the ideas to overcome this problem is the convergence of the Cloud with Grid along the line of inter-Clouds federation. It must be taken into account that one of the most important integral element in this convergence procedure is the problem of resource sharing, which contains the issues related to information dissemination, resource matching and discovery.

The resource scheduler in the Cloud (in case of computing Cloud) receives the user request, which includes a set of required hardware to build a customized computing system. Afterwards, the scheduler starts to lookup a matched set of physical or virtual resources. If the required resources are not available in the system, the Cloud service provider (system manager) has to create virtual resources (resource/service provisioning) which are precisely matched to the request. So despite the Grids super-schedulers (Grids controllers), Clouds schedulers are responsible for both resource discovery and resource provisioning. In Grids, each Grid site may have a super-scheduler (or global/meta scheduler) which is responsible to allocate jobs received from clients to the global/local resources found by resource discovery.

Clouds are based on a Service Oriented Architecture (SOA) where each service is accessible through a broker. The clients submit their requests, containing the required QoS levels for the desired services to the brokers. Consequently, the brokers proceed to find and allocate the best matching service provider for the requested services with certain level of QoS. There are a significant number of commercial brokers such as Cloudswitch [137], Deltacloud [138–140], Eucalyptus [141] and Elastra [142, 143] which already are used in different Cloud systems. However, there is still no standardized method to map Cloud services to the clients requests, based on QoS level.

Considering the administrative boundaries of the Clouds and the fact that Clouds allow resources that can be elastically divided and reassembled to meet the end users requirements, the concept of resource discovery in Cloud differs from Grid. Here it has to support scheduling and resource management functions such as resource provisioning, resource brokering, resource mapping, resource allocation, resource modeling, and resource adaptation. Resource discovery, hence, is specially emphasized when we move toward inter-Clouds approaches or Self Organized Clouds (SOCs). Each single Cloud provider or individual host are required to autonomously locate and discover a set of qualified volunteer computing resources in the network for its job's execution via different types of querying strategies. Since *i*) we want to focus on resource discovery and we are not going to discuss in depth other relevant resource management aspects (like resource provisioning) in Cloud computing (there is a comprehensive survey to discuss the resource management issues in Clouds in [144]); and *ii*) due to the point that most of resource discovery solutions for Grid and P2P are also applicable to be used in Clouds; we focus in

this chapter on the Cloud architectures and discovery solutions which have specially designed for multi-Clouds environments such as federated Clouds, hybrid Clouds and self-organized Clouds.

The increasing number of Cloud services along with the massive amount of global users, and inherently limited scalability of the current single provider Clouds, became the reason to the development of multi/many providers Clouds. In such an environment, called a federated Cloud, resource discovery becomes a very important issue to address the federated placements which refers to the process of clarification of the most appropriate cluster to use for a particular application workload, considering the point that all Clouds are not equal in terms of capabilities, performance, QoS, availability warranties and cost. Along this line, there are a number of resource-centric IaaS providers such as Eucalyptus, OpenNebula [145] and Nimbus [145] that provide the clusters of virtual machine hosts as IaaS resources. They offer the list of their resources types containing the prices and the detail of capabilities for each particular resource type.

RESERVOIR [146, 147] introduced the notion of federated Cloud. The federated Cloud contains several single provides Clouds joined by a mutual corporation agreement which makes inter-Cloud computing and cross-Cloud migration feasible in a cost-efficient manner. Providers that have excess capacity are able to share their extra available infrastructure resource with the federation members that need those resources to overcome the problems such as resource limitation and over-provisioning. RESERVOIR is an open federated Cloud computing model, architecture, and functionality which aims to deal with scalability, complexity and interoperability in multiple Clouds environments.

In the RESERVOIR Cloud model, two or more Cloud providers join the system to create a federation of the Cloud providers where various Cloud services or resources dynamically can be managed and controlled together. Logical services/resources are represented and encapsulated in a Virtual Execution Environment (VEE). VEEs are hosted on top of physical resources and virtual application networks which are represented as Virtual Execution Environment Hosts (VEEHs) managed by a Virtual Execution Environment Management System (VEEM). The VEEM is responsible to perform the management, deployment, and migration of VEEs on top of VEEHs through the utilization of OpenNebula. Moreover, the service manager component is responsible for instantiating services and manage the SLA. RESERVOIR provides facilities to describe and specify the application configuration, which results in defining a service by a set of information such as the specification of the virtual machine, application configurations, and deployment settings. RESERVOIR attempts to maximize system flexibility to address the different requirements and policies from different infrastructure resource providers, by supporting dynamically-pluggable policies to measure and calculate the placement. Different policies define different utility functions which have to be optimized. For example, a load balancing policy aims to distribute the Virtual Machines (VMs) equally between physical resources, while an energy saving policy tries to minimize the number of physical resources for VM allocations; thus, to reduce energy consumption, the unused machines can be turned off. The RESERVOIR Cloud model enable the individual Cloud providers to have a focused view of resources, while fully preserving the individual autonomy of the providers in making technological, policy and management decisions, by leveraging and combining the advantages of virtualization and embed autonomous management. This helps the resource/service providers to completely describe and define their resources that helps to implement efficient and accurate resource discovery approaches along the line of Cloud computing requirements.

Another example of resource discovery, for multi/many providers Clouds, is presented

in Wright et al., [148]. It deploys a two-phase constraints-based resource discovery solution which uses a software abstraction layer to discover the most suitable infrastructure resources in a multi-provider Cloud environment for a given application request. In the first step, the discovery system identifies a set of potential infrastructure resource candidates who can satisfy the application requirements in terms of quality of service and in the second phase, an appropriate heuristic method depending on the application preferences (for example, some application may prefer to use a cost-based heuristic while others can prefer the performance-based heuristics) is used to select the best matching candidate among the initial set of discovered resources. The proposed model shows a dynamic flexibility to choose the resources based on the application requirements and preferences.

2.3.3 Ontology and Resource Description

In the computing environment there are several different entities (e.g., resource providers, resource requesters and directory agents) that are seeking a common purpose of resource sharing. For this, at least they should have a common understanding of resources, which requires shared definition of resources in terms of ontology (description and definition of resources) in order to be able to work to each other and reach an agreement to perform resource sharing through discovering, mapping, allocation and invocation of resources.

Resource discovery and allocation can be simply defined as the process of finding and mapping computing applications (or application segments) to the appropriate available computing resources in an efficient manner, where, in each application query, the discovered (selected) resources are the best matches for the application requirements while the resources in the system are efficiently utilized. For this purpose, it is required to abstract the complexities of the system components such as computing applications (application description) and hardware computing resources (resource description) through a consistent methodology and language which describe their characteristics, requirements and capabilities. In fact, the resource capabilities can be described and derived from hardware descriptions. According to the application description the discovery system must be able to exploit the application resource requirements in order to generate the proper query. In the next step the query explores the network to find the best matching resource through the evaluation of the resource description of each individual resources.

Distributed resource discovery in a large-scale system requires a scalable and fully expressive resource description, which contains a required level of information details in terms of computational (e.g., processor description) and communicational (e.g., topology description) properties and behaviors. Most of the resource discovery behaviors and operational characteristics are under the influence of the way we abstract, organize and distribute the individual resource information. For example, in a common P2P system, for the purpose of resource discovery, each one of the peers has to get information from other peers and disseminates the information to others through neighbor peers. In this case, it is important to consider the point where the resource information has to be distributed and get balanced between peers. It helps to improve the scalability of the system when the number of peers grows increasingly. Moreover, resource description has direct impacts on the indicators such as discovery latency (response time), accuracy of results (rate of false discovery), discovery traffic overload, etc. We categorize the approaches for resource description in two groups: attribute-based and semantic-based schemes.

The attribute based schemes define the resource characterizations through a set of (attribute,

value) pairs. Depending on the level of information details and the different storage, retrieval and distribution mechanisms these approaches are able to provide a scalable distribution of resource information. However, the attribute-based description models are facing some challenging issues such as providing appropriate support for dynamic and collective attributes [149]. Dynamic attributes are changing frequently, which results that the above mentioned description models becomes unappropriated to store their values due to expensive updating and maintenance cost. On the other hand, since all resource properties are not independent and they are related to each other in some aspects (depending on the level of abstraction), it would be more complex and difficult to exploit all resource properties and capabilities individually in the form of (attribute, value), without leading to information redundancy and inaccurate resource description.

XML, Web Services Description Language (WSDL) [150], Ontology Web Language (OWL) [151], GSDL [152], OASIS Universal Description, Discovery and Integration (UDDI) [153–155] and [156] are some examples of attribute-based resource descriptions which have been used in many resource discovery solutions specially in web-based resource discovery protocols. WSDL is a set of instructions to describe the behavior, characteristics and formats of services, particularly for web service providers. UDDI uses a XML-based repository to provide policies and standards for service discovery which facilitate the process of resource advertisement and service publication. Tutschku et al, a recent work [156], is proposing a resource description method based on the capabilities of on-board Linux tools for describing resource utilisation in cloud networking and NFV infrastructures. It aims to provide a description approach for dynamic attributes such as CPU load and utilization. However, the approach is not general (it is based on linux) and is limited by very abstract description of resources and also the description is restricted to a very few number of predefined general attributes. In addition to the attributed-based description languages, there are a number of recent attributed-based (resource information) encryption methods in the current literature including [157–162] which are mostly application-oriented. And in fact, they are not really flexible and even efficient to be used for different applications.

Semantic-based description models are the alternative approaches which focus on the overall collaborative description of the resources. These approaches are appropriate to describe all system resources where all the possible collaborative system and resource properties and behaviors (e.g., the structure of the processor or memory architectures) are precisely described, but we must take into account that these approaches might not be scalable in terms of distribution of the resource information. The works in [163–166] are examples of semantic-based schemes which use functional languages to describe hardware resources. The use of higher-order functions allows the composition of arbitrarily complex structures in a clear and concise way. The strong type system of most functional languages also ensures the soundness of the composition of the different hardware components. Functional resource description models focus on capturing the structure as well as numerical properties of hardware resources. Resource descriptions themselves are functions, capturing the fact that behaviors/capabilities relevant for a resource can change under certain circumstances. Additionally, functions can be used to concisely and clearly capture complex, parameterizable collaboratives. For other examples of semantic-based ontologies we can mention Resource Description Framework (RDF) [167–169], Ontology Inference Layer (OIL) [170], DAML+OIL [171–174] and DAML-S [175].

RDF is a type of ontology/knowledge representation approach, which is a primitive language providing a binary relation of classes and properties supporting all kind of range/domain constraints and sub-property/sub-class relationships. It is a powerful and more expressive

language to describe resources in sufficient details. OIL is an extension of RDF while DAML+OIL is an ontology representation language derived from the work under DARPA's Agent Markup Language (DAML) [176] (based on the OIL language). In comparison to OIL, it provides a larger interoperability on the semantic level through extending the OIL and RDF basic primitives along the way to provide an ontology-based description language with better expressiveness and capability for inference creation. DAML-S is another work under the DAML program which provides a set of principles, basic concepts and relations to describe and declare services/resources, by implementing the ontology structuring mechanism of DAML. Each DAML-S service/resource is characterized in three types: service/resource model (process model), service/resource profile and service/resource grounding. These describe respectively the service functionalities (i.e., the service composition and the service behaviors on the run time), the service capabilities (i.e., the required information for resource/service discovery), the service access (i.e., the service address or the required information for service/resource invocation). In fact, resource model and resource grounding provide the required information for interaction between resource providers and resource requesters, while the resource profile facilitates the process of matchmaking for resource discovery. There are several proposals such as [177, 178] that utilize DAML-based resource description for resource discovery.

The Lexical Bridge [179] is a recent work which proposes a methodology to translate meaningful information in natural language sources into a standardized, structured knowledge representation for the purposes of semantic normalization, integration, analysis, and reasoning. However, it is a very general work and in fact doesn't provide a resource description language (for the purpose of resource discovery in the distributed system) a complete resource description language, rather, it aims to build "lexical bridges" (LBs) in order to fill the gap between the natural languages and their ontology representations. The authors in [180] have highlighted that a scalable resource description and information exchange (in terms of distribution of resource information between Clouds) is an important requirement for sharing heterogeneous Cloud resources among federated Clouds. However, the work presented in this paper, a semantic based resource description model for inter-clouds, does not really provide a scalable solution for distributing all details of resource information in the entire system. It focuses on providing a scalable information exchange method between multiple clouds, where each cloud centrally manage its own resources.

Considering the above approaches, any optimum resource description model for resource discovery must be designed in such a way that it takes the concerns of both approaches (attribute-based and semantic-based): capturing individual and collaborative capabilities of resources while still allowing for scalable distribution of this information [180]. Consequently, a merger of the two above abstraction concepts should be pursued.

2.3.4 Clustering Approaches

Clustering is the process of altering the network topology in order to increase the overall system performance. Nodes and resources are grouped in clusters, which share common specifications, properties, operations or behavior. Overlay construction and clustering enhance the efficiency of the query/information management and resource advertisement for the resource discovery systems. It must be taken in account that the system overlays have impact on a set of important discovery performance factors such as scalability, efficiency and even reliability and dynamicity. Clustering approaches may provide some inherent features like locality-awareness, which could bring benefits to enhance the discovery mechanism.

Generally, we can classify the clustering approaches (see Table 2.4) for resource discovery in three main groups: quantity based (none-content based), quality based (content based) and hybrid clustering. In quantity based clustering, overlay construction involves the methods to organize nodes/resources in the groups without considering the content (for example, in terms of attributes, features, properties, behaviors) of nodes/resources. For this purpose, the focus might be on some performance related issues such as load balance, maintenance, self-organization, stability (churn and fault tolerant), while the overlay must be constructed along the way that finally satisfies all the requirements of a resource discovery protocol which aims to run on top of it.

Table 2.4: Examples of clustering approaches for resource discovery (OA: Overlay Architecture).

OA	Mechanism	Clustering	RD Examples
Semantic-Aware	Nodes/resources with similar contents are organized in the same cluster	Content-based Clustering, SON [98]	ERGOT [97], The works in [52]
Proximity-Aware	Physically nearby nodes/resources are organized in the same cluster	Content-based Clustering	CycloidGrid [64], TriPod [63], PIRD [65], PIAS [181] and ASTAS [182]
QoS-Aware	Nodes/resources with similar quality of service are organized in the same cluster	Content-based Clustering	CycloidGrid [64]
Load-Aware	Nodes/resources are organized in a particular overlay (e.g, DHT ring) in order to equally distribute the loads (in terms of resource information or query workloads) among nodes to enhance the overall system performance.	None-Content based Clustering, P2P overlay network (e.g., DHT based P2P)	PIAS [181] and ASTAS [182]
Super-peer	Nodes/resources are organized in a particular overlay (e.g, tree, hierarchy) in order to enhance the overall system performance.	None-Content based Clustering, None-DHT based P2P Overlay	The works in [58, 101, 102]
Tree	Load balanced Tree Overlay	Tree	MatchTree [62]

Quality based clustering approaches organize the groups based on the content similarity of each individual nodes/resources in different aspects. According to the different views and definitions of the concept of the content similarity, there are several well-know approaches for content clustering which shapes the discovery mechanisms in the directions. Proximity-aware [183–187], semantic-aware [188–192] and QoS-aware [193–195] resource discovery are the sample applications of content-based clustering.

QoS is one of the most important aspects of resource discovery systems in large scale environments, specially in Grids and Clouds [196]. It must taken in account that the QoS concept is not only limited to network capabilities like network bandwidth, rather it is extended to all kind of computing and storage capabilities. In fact, when a task with QoS conditions is submitted to a Cloud or Grid system, it would be necessary to negotiate a SLA in order to guarantee the quality of the requested service according to the QoS conditions. Accordingly, resource reservation is one of the best mechanism benefiting from QoS. As an example of QoS-aware resource discovery, the MDS Globus provides this kind of resource reservation capability [24]. However, this reservation has a poor efficiency to guarantee the bandwidth for the network links which results that the system would not be able to guarantee the reservation of the processing cycles for the application segments that communicate over those network links. Currently, most of the Grid/Cloud systems provides only a partial solution for QoS,

due to the point that the majority of the processor operating systems do not provide precise performance guarantees. Partial QoS solution means that the system provides capability to specify QoS conditions at the time of job submission while it does not have support for resource reservation mechanism.

2.4 Design Aspects

In order to design a resource discovery approach, it is required to specify which design choices to make for different parts of the discovery process. A resource discovery system, first, must be initialized through using a bootstrapping mechanism. We must also specify an overall discovery strategy. According to the overall strategy of the discovery, an adequate search method must be applied for directing and processing the queries in the system. In this way, it might be necessary to specify a method for the propagation of queries in the system. Furthermore, it is important to provide a strategy for information delivery by queries (each query may contain a certain types of information which are delivered between different entities in the system). A discovery approach also needs to provide a mechanism for query termination, otherwise, a query may search forever or multiple computations of a single query might happen in different places of the system, due to the same query arrival through different discovery paths.

2.4.1 Bootstrapping

Bootstrapping is the process that occurs when a new element joins the discovery systems. The new node may not have information about the other nodes/resources in the system overlay. Thus, in the first step it is required to discover a node (bootstrapping node) that already belongs to the system overlay, or at-least has some initial information about the system (e.g., the address of the resource provider nodes, the super-peer nodes, the directory agents, the server nodes, the neighboring nodes or even the multicast address of the group). In other words, the new nodes will join an existing system through asking a bootstrapping node for a list of already known nodes to which it uses later to perform a resource query.

There are several mechanisms [197] for bootstrapping. In a small network, the new node can announce its presence through conducting a simple flooding. Since, flooding is not efficient in larger networks, multicasting can be used instead of flooding. In centralized structured systems, it would be easy to use a service like Domain Name System (DNS) [198], thus, the new nodes can easily use the DNS to get the server address. In traditional service discovery systems [199] like SLP [200], both service and user agents try to find a directory agent (Active method) for either service-announcement or service discovery. The Passive method lets strategic nodes such as directory agents or bootstrapping neighbor nodes to periodically announce their information in the system through multicast or even broadcast. Generally, in decentralized and in unstructured systems, the new node asks the bootstrapping neighboring nodes for information about the system overlay. The bootstrapping nodes can be either a central registry server or any node that already belongs to the network. The problem is that, the central registry server potentially becomes a single point of failure and, in the lack of bootstrapping node in the neighborhood, a non-bootstrapping node may not provide enough information to determine the positioning of the newcomer nodes in the system overlay. Other possible techniques for bootstrapping in P2P environments can rely on mechanisms to

determine the initial neighbors, such as caching (i.e., using the previously stored list of active nodes), and local network broadcasting [192,201].

2.4.2 Discovery Strategies

Discovery strategy specifies the overall approach for sharing resources information across the system. A search method for handling queries can be designed with respect to an overall discovery strategy. In general, there are three types of discovery strategies (i.e., modes of discovery operation): Reactive (pull-based), Proactive (push-based) and Hybrid.

In the Reactive mode, the requester creates a query and sends the query to either a resource provider or a resource directory agent. Since the destination of the queries is already known, it is not necessary to flood the system. In the case that the directory agent or resource provider does not exist, the search range can be expanded by increasing the number of hops. Therefore the requester would be able to find the nearest resource providers or directory agent. This can be done by using different propagation mechanisms such as unicast, multicast or broadcast. The requester can simultaneously send the query to one or multiple resource information providers. The Reactive mode avoids flooding by eliminating advertisements. The drawback is that query messages take longer to find resources, due to the blind nature of the search mechanisms.

In the Proactive mode, the information providers (resource providers or directory agents) periodically advertise their resource information to the environment, therefore every node in the system can update their information. Using this information helps the requesters to locally resolve most of the queries, however, it might be possible that, in a given time frame, the local resource information is not up to date and leads to invalid discovery results, decreasing system accuracy. On the other hand, system maintenance based on periodical updates (i.e., broadcasting the resource information to other nodes) is not cost-efficient and creates large amount of network overhead.

The Hybrid mode uses the advantages of both Reactive and Proactive mechanisms by combining and aggregating these mechanisms for all kinds of discovery components such as resource providers, resource requesters and directory agents. As an example of Hybrid mode, we can mention cluster based resource discovery protocols, which generally use Proactive mode within the clusters (intra-cluster discovery) and invoke the Reactive mode for searching between clusters (inter-cluster discovery). When the number of resource requesters are significantly more than the number of resource providers the Proactive mode provides better performance in terms of latency and overhead while for the environments which has large number of providers with few requesters the Reactive mode is more efficient [202,203]. However, in both cases, the Hybrid mode provides better performance [204].

2.4.3 Search Algorithms

Search techniques are the most challenging part of resource discovery systems. There are various techniques to discover and locate resources in the network, which are differentiated based on their algorithms and the target usage environment [14, 15, 205]. For example, in a small network, with limited number of resources, no complex search method is required. A node can simply discover other resources by using basic broadcasting or multicasting. Directory-based or centralized systems with limited number of servers also do not require a complex propagation method for querying. However, in large distributed networks, such

as unstructured peer to peer overlays, to support complex or free-form queries, appropriate search techniques have to be applied and integrated with the query propagation methods to increase efficiency and scalability. In this section, we discuss some of the most well-known algorithms which are used for resource discovery in large scale systems. For this purpose, we classify search methods, in the literature, along different dimensions such as informed vs uninformed, synchronous vs asynchronous, deterministic vs none-deterministic and bio-inspired vs none-nature inspired (see Figure 2.13).

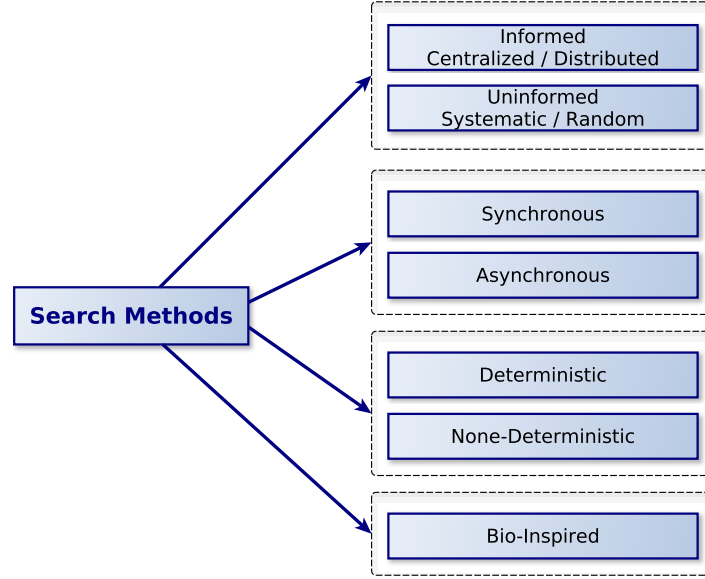


Figure 2.13: Search algorithms for resource discovery.

2.4.3.1 Informed vs Uninformed

According to the system overlay and the distribution of the resource information, we can classify the search methods in two groups which are informed and uninformed (blind) search methods [206]. In the uninformed search approach (blind search), the sender node knows nothing about other nodes and resources in the network, while in the informed search approach, each requester or intermediate node at least has some prediction about the location of the resources requested. Uninformed methods can be either based on systematic or random algorithms, where in systematic approaches, almost the whole or part of the search tree or graph is explored. In random based algorithms, there is an option to reduce the size of exploring space by randomly choosing the next node or invoking some probabilistic technique to disseminate the query.

The underlying concept of most of the blind (uninformed) approaches is based on the reduction of communication complexity (through resource information replication) by limiting the spread of the queries through mechanisms such as setting small or dynamic time-to-live values for query dissemination, forwarding the queries to only a random chosen subset of nodes or by implementing different strategies for resource information replication like path replication and uniform distribution across the network. On the other hand, limiting the spread of the

queries may increase the time complexity of the search method, which leads to slow resource discovery. The time complexity may be improved by query replication. Following, we discuss some of these approaches in more detail.

ALG-Flooding [207–209], Breadth First Search (BFS) [210–212] and Depth First Search (DFS) [213–215] are the most well-known systematic search methods. Moreover, there are several other systematic and random search methods such as Depth Limited Search [216], Iterative Deepening [217], Uniform Cost Search [218], Random Walk [219–221] and Gossip based search methods (e.g., Newcast Gossiping [222], works in [223, 224]). Referring to the aforementioned solutions, there are various resource discovery protocols that integrate them with their own query propagation methods such as Probabilistic Flooding [225], Forwarding Protocols (e.g., Selective Forwarding, Intelligent Forwarding and Probabilistic Forwarding) [226, 227], Gossip-based Probabilistic Forwarding [228–231], etc.

In BFS, the search algorithm is deterministic where the algorithm ensures that if the resource is already located in the system, it will be found as the result of resource discovery. The search process is initiated by the source requester, which propagates the query to all of its neighbors. Consequently the receiving nodes forward the query to all of their neighbors, except the one from where the original query came from. Additionally, intermediate nodes avoid further flooding of a query, which has already arrived and processed earlier. The query terminates either if the query is resolved or if there is no edge that query has not passed. The weakness of this method is its huge discovery overhead to find the required resources (due to the costly nature of the flooding) (see Figure 2.14).

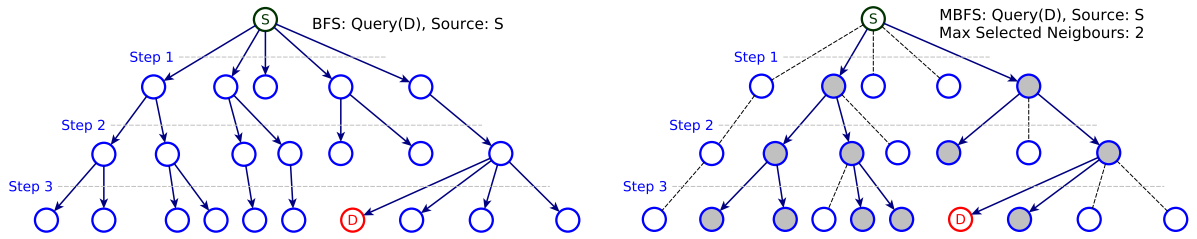


Figure 2.14: Examples of query processing using BFS and MBFS.

Modified Breadth First Search (MBFS) [232] is a variation of BFS in which the nodes only select some of their neighbors for query forwarding (see Figure 2.14). In comparison to BFS and ALG-Flooding, it reduces the number of transmitted messages to resolve a query, but it still suffers from large traffic overhead, which comes from its flooding nature.

In the Standard Random Walk solution [233], the query message (walker) is forwarded to a randomly chosen neighbor at each step, until the required resource is found. It leads to a significant reduction in the message overhead, but the solution is slow and requires a large number of hops to resolve the query (see Figure 2.15).

The K-Walkers Random Walk solution [233] reduces the discovery latency (related to the number of hops) in the standard random walk by increasing the number of walkers. The requesting node sends out the query messages (walkers) to randomly selected k neighbors. Consequently, in each round, the receivers (intermediate nodes) forward the query messages to a randomly chosen neighbor. The walkers proceed to walk to the next nodes until the required resource is found or the query is expired, which happens after a certain number of hops (see

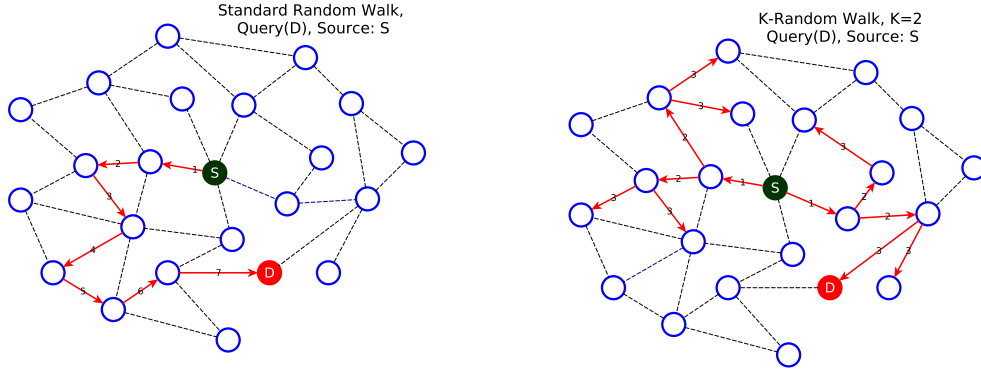


Figure 2.15: Examples of query processing using Standard Random Walk and K-Walkers Random Walk.

Figure 2.15).

Query termination can be based on either the TTL method or by an explicit verification method. The advantage of this algorithm is its significant overhead reduction in comparison to other pure blind search methods, while it is faster than standard random walk. A common problem affecting most of the blind search algorithms, such as the random walk search, is message duplication, which happens in the case that a node sends duplicate query messages to other nodes regarding the same query. This creates a significant unnecessary traffic over the network. Moreover, the random walk approach is none-deterministic, which means its success is not always ensured, and the rate of successful discovery would be small for the rare resources.

The informed search methods can be classified in two groups: mechanisms that set specific nodes to act as index nodes (index servers), and mechanisms where indices are distributed between all the nodes (i.e., each node can be an index node). The approach employing specialized index nodes has been used in most of the resource discovery approaches such as Napster, Kazaa and JXTA, whereas they utilize a set of servers or super-nodes to maintain the extra information about their sub-nodes or other super-nodes/servers. However, these mechanisms are not scalable, and they are vulnerable to attacks due to their centralization of indices in a small sub-set of the nodes. Along this way, the solutions based on distributed index nodes such as intelligent search [232], bloom filter based search [234], local indices based search [235], routing indices based search [236], dominating set based search [237] and adaptive probabilistic search [235] are more scalable and reliable.

2.4.3.2 Synchronous vs Asynchronous

We can model the resource discovery problem (specifically in a decentralized fashion) using concepts from graph theory, where the computing nodes and communication links, are represented by the graph vertexes and graph edges. At the initial state, some of the nodes may know about the resources of some other nodes, for example, each node possibly knows about a set of initial neighbors. Furthermore, each communication link between two nodes demonstrate that those nodes know about each other. These relations and the environment can be presented as a weakly connected graph (uninformed model). The resource discovery search

algorithms can be defined as the solutions that are able to converge the above mentioned weakly connected graph to a complete graph, where all the computing nodes know about resources of other nodes in the network. Along this line, to provide an efficient search algorithm, the network communication complexity, and the number of required rounds for graph conversion (in terms of time, hops or cycles), must be low. Based on the above abstraction, we can classify resource discovery algorithms in two categories: synchronous and asynchronous methods (see Figure 2.13), where these later can be either informed or uninformed, deterministic or none-deterministic.

In synchronous methods [117, 238–241], the search algorithm proceeds synchronously, in parallel rounds, where each round is defined as the time required for each node in the network to communicate with one or more other nodes, learning about them and gathering information. Asynchronous methods [242–244] are based on the asynchronous communication model, where each node can send a message in arbitrary size to any of its neighbors in a way that the outgoing message eventually arrives in the destination node after an unbounded finite time. Moreover, unlike the synchronous method, there is no assumption to start the algorithm simultaneously in all the nodes and the receiving nodes simply follow the First-Input First-Output (FIFO) model to serve the requesters.

Harchol et al., [117] discussed some well-known natural algorithms such as ALG-Flooding [245], Swamping [245, 246], Random Pointer Jump [245] and presented Name-Dropper [117] to perform a synchronous resource discovery. Table 2.5, compares these algorithms to other approaches with respect to three performance measures: time complexity, pointer complexity and message complexity. For a discovery request: the time complexity is the number of time steps taken; the pointer complexity is the number of nodes/pointers (e.g., machine addresses) passed; and the message complexity is the number of messages sent. In the following, we discuss these algorithms in more detail.

Table 2.5: Comparison of some well-known synchronous search algorithms for state discovery.

Search Algorithm	Time Complexity	Communication Complexity	
		Pointer Complexity	Message Complexity
Absorption [247]	$O(\log n)$	$O(n^2), O(n^2 \log n)$	$O(n), O(n \log n)$
Kutten & Peleg [248]	$O(\log n \log^* n)$	$O(n^2 \log^2 n)$	$O(n \log n \log^* n)$
ALG-Flooding [117]	$d_{initial}$	$\Omega(n \cdot m_{initial})$	$\Omega(d_{initial} \cdot m_{initial})$
Swamping [117]	$O(\log n)$	$\Omega(n^3)$	$\Omega(n^2)$
Random Pointer Jump [117]	$\Omega(n)$ in worst case	$number.of.cycles \cdot m_{initial}$	$number.of.cycles \cdot m_{initial}$
Name-Dropper [117]	$O(\log^2 n)$	$O(n^2 \log^2 n)$	$O(n \log^2 n)$
Fast-Leader [249]	$O(\log^2 n)$	$O(n^2)$	$O(n \log^2 n)$

When using ALG-Flooding algorithm, each node only communicates with its (manually configured) initial set of neighboring nodes, thus the new added edges are not used for communication. In each round, every node sends updated information, including the new nodes that joined recently (see Figure 2.16). The number of required rounds to converge the graph to a complete graph depends on the diameter of the initial graph. Furthermore, in the ALG-Flooding algorithm, every pointer information must be transmitted over every edge in the initial graph. Thus, the network (or communication) complexity of the algorithm (in terms of pointer and message complexity) depends on the number of edges in the initial graph. Using flooding-based methods for resource discovery in environments like large scale Grids will reduce the overall system performance, since flooding increases network traffic congestion.

Swamping is a flooding based search algorithm with the difference that, each node may send requests to all of its neighbors, and not only a fixed initial set of the neighboring nodes. Since nodes transfer the current set neighbors to the target nodes using the request messages, the neighbor sets are dynamically changed and updated, which leads to a very fast search algorithm. However, this speed is achieved at the cost of wasting communication bandwidth, because many nodes will receive information about nodes that they already knew about (i.e., they have them in their local set of neighbors). This increases the network communication complexity very quickly, which reduces the algorithm's performance (see Figure 2.16).

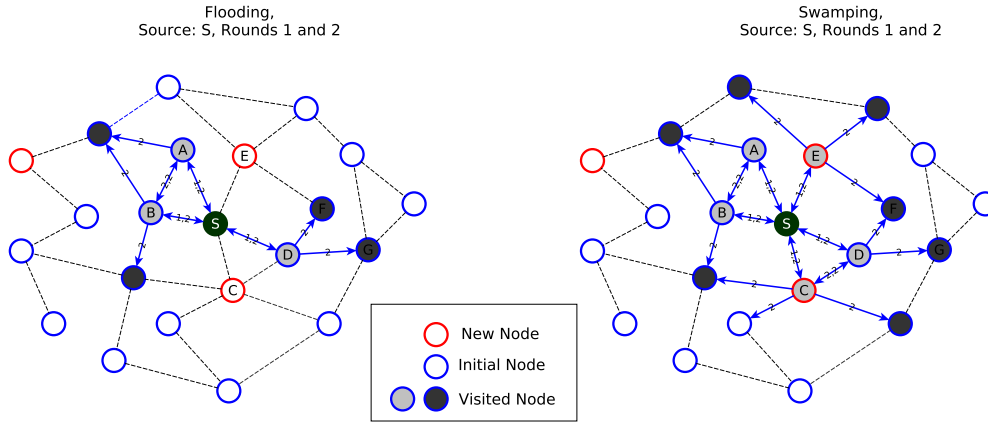


Figure 2.16: Examples of information dissemination using ALG-Flooding and Swamping.

Concerning the problem of the communication complexity in the Swamping algorithm, the Random Pointer Jump algorithm proposes that in each round, every node can only send request to a single random neighbor. Upon receiving the request, the receiver node will send the information of its neighbors to the node made the request (see Figure 2.17). The solution results in lower communication complexity (i.e., the average number of transacted messages and visited nodes to resolve a query will be decreased). However, the resource graph in the initial stage must be a strongly connected graph (informed model), otherwise it will never converge to a complete graph. Moreover, even for the strongly connected graph, there is a high probability that the number of rounds required to achieve a converged graph will be large. The Random Pointer Jump with Back Edge [250] attempts to address this problem. Using the Back Edge algorithm, when a node receives a request, it adds the information of the sender to its neighboring set. Afterwards, it sends its neighboring information to the requester node. This enhancement increases the performance of the Random Pointer Jump algorithm, but the main drawback is that there is still no guarantee that the resource graph converges after a specified number of rounds.

The Name Dropper is an enhancement to the Back Edge algorithm. In each round, each requester sends its neighboring information within the request to a single random neighbor. The receiver will merge the received neighbor information with its own set of local neighboring information. The rest of processes will be performed just like the Random Pointer Jump with Back Edge algorithm (see Figure 2.17). Name Dropper achieves better performance by providing low communication complexity and low number of rounds required. The drawbacks are that the algorithm is not deterministic and also it is not able to determine its termination

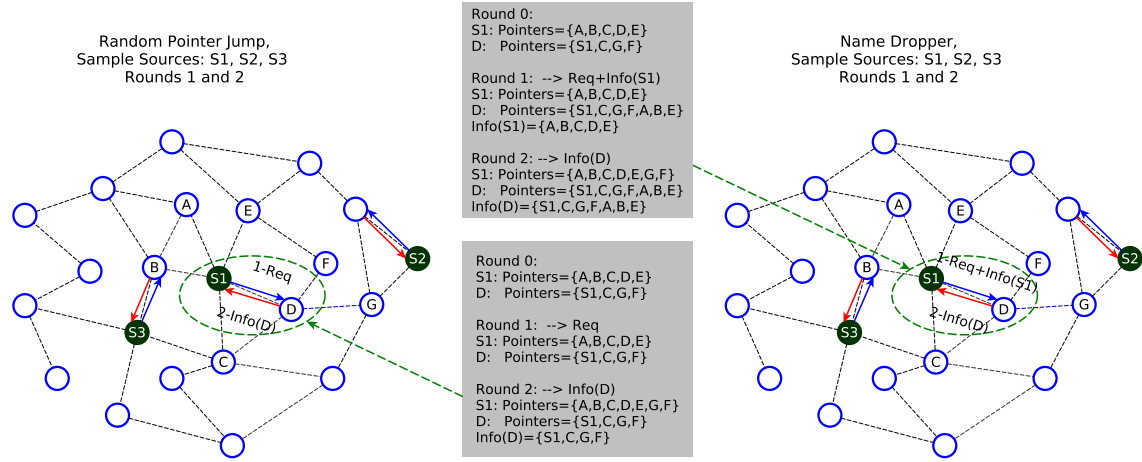


Figure 2.17: Examples of information dissemination using Random Pointer Jump and Name Dropper.

point, unless the number of nodes in the graph is defined in advance. The Fast-Leader [249] algorithm has been proposed to overcome these drawbacks. The algorithm includes two steps. In the first step, the initial strongly connected graph will be converged to a star graph, where a central node knows about all other nodes. This is achieved through a leader election algorithm. In the second step, the star graph will be converged to a complete graph by using broadcast propagation, where the central node broadcasts all the information to all other nodes. In opposition to Name-Dropper, Fast-Leader is a deterministic algorithm, which its runtime (number of rounds required to resolve a graph) and communication complexity match those of Name-Dropper. Furthermore, the convergence detection approach (i.e., recognizing the termination point of the algorithm, which is the time that the initial graph has converged) is an inherent feature of Fast-Leader, and the algorithm halts as soon as convergence is obtained. However, unlike Name-Dropper, Fast-Leader is only applicable to resolve the strongly-connected initial graphs. Another shortcoming is that the algorithm is more resource consuming regarding the memory usage and computation cycles per node in each round.

The above mentioned natural synchronous search methods have been used in many resource discovery approaches for different environments like Grids. However, these algorithms in general have some critical weaknesses. They do not support (or weakly support) dynamic environments considering different dynamicity issues in terms of rapidly and frequently changing network topologies (i.e., node arrival, node departure, node failure, resource expiration, or adding new resource) and resource attributes (e.g., CPU load, memory utilization). For example, after running the algorithms (by converging the system resource graph to a complete graph) each node will have the information about all of the other nodes in the network. However, this would be useful only for static information, with the additional assumption that the system is static and reliable, which is far from real. In other way, each node needs to iterate the search procedure to update its information, which is not efficient. Furthermore, using periodic updates also creates a huge amount of network overhead.

Essentially, in a real environment, it is not necessary that all nodes in the system simultaneously know about others, due to the fact that the transaction and maintenance of large

amount redundant information over the network is a waste of resources. For example, a resource discovery approach might be interested in only discovering the closest resources inside the border with a specific number of hops or latency.

2.4.3.3 Gossiping-Based Approaches

In gossiping, two random, non-requester members of a group repetitively talk (exchange information) about a query, generated by a requester in the system. Gossiping might be strongly related to epidemics, by which a disease is spread by infecting members of a group, which in turn can infect others. Epidemics refers to information dissemination between randomly selected members where the information (i.e., the “disease”) can be communicated among an expanding number of members [251].

In order to solve the problem of dynamic load balancing, Gossiping based search methods (Epidemic Query/Information Disseminations) provide alternative mechanisms to perform querying in the distributed environments whereas they have shown efficiency in regard to information dissemination. They are based on flooding, but in a manner that generates bounded worst-case network loads. In general, to find a match for a resource request using a Gossiping protocol, the requester node starts to gossip about the required resource with some other randomly selected nodes (e.g., picks one of the neighboring nodes), thus the query (required resource) will be known for both of sender and receiver in the first round. Consequently, in the next rounds each node that already knew about the query repeats the process by gossiping to another random selected neighbors (i.e., periodical querying/updating respectively by exploiting either push based or pull based approaches) until the query is matched or the query is over aged. Nodes also gossip about the best match after they find the matches, (i.e., the best matches will spread through the network). Gossiping based solutions might be considered as a type of epidemic protocol, due to the fact that a node that gossips a query or a resource information can be seen as it infects its neighbor. The advantage of Gossiping is its powerful capability for disseminating and delivering contents to (almost) all destination peers with a very high probability [226, 252, 253]. Thus these solutions may become fast, scalable and load balanced, since they provide a statical guarantee on the number of nodes involved during the query/information dissemination process. Moreover, they are resilient to topology changes and churns due to their decentralized nature. The drawbacks come from the fact that Gossiping based search methods are more concentrated on the load balanced query/information dissemination and they do not provide approaches in other aspects, such as the way to route queries, or mechanisms to match a query to a resource description. The works in [228, 254–257] are some examples of gossip based resource discovery protocols.

2.4.3.4 Bio-Inspired Approaches

Bio-inspired search methods (see Table 2.6) are based on biological systems, which have native capabilities to exhibit autonomy in various levels, ranging from molecular independent self-management and self-organization of organisms, to the large scale adaptation of animal in communities and colonies. Other than autonomy, these systems have some other desirable inherent properties such as scalability, adaptability and robustness, which create motivation to apply bio-inspired search methods to discovery systems. A bio-system contains bio-organisms interacting in a bio-environment where bio-organisms are the self-organized autonomous entities without any central controller, and the bio-environment provides a medium for communication

and interacting between bio-organisms.

Table 2.6: Summary of bio-inspired search methods (SA: Search Algorithm).

SA	Description
Bee Colony [258–262]	A swarm based optimization algorithm which is inspired by foraging mechanisms of honeybees. The scout bees stochastically search (global search) the new food sources, upon finding a new source, they will transfer the flower information by performing waggle or round dance. In other side, the worker bees evaluate the quality of the new discovered flower sources through observing the dances and choose the elites (best sources) for foraging. Bees must avoid overcrowding and keep diversity, for this reason they scatter in the proximity around the elite flowers (local search) while in the same time the scout bees start a new iteration of the global search. Bee Colony Algorithm (BCA) divides the search process in two steps, parallel implementation of the global search (parallel random search in variable space by scout bees) and the local search which is the local improvement of the current elites by worker bees. The elite selection mechanism in each iteration compares the quality of the discovered solutions from global search and local search with the quality of the elites in the last iteration.
Ant Colony [263–266]	Ant Colony Search is a meta-heuristic search method based on Ant Colony Optimization (ACO) which is inspired from the behavior of the real ants searching their environment to find the food sources. The ants (queries) start their search by randomly parallel scattering around the nest. The successful ants whose found the food sources (resources) will return to the nest using their memory and mark the (successful) path (between nest and food source) through emitting a chemical substance called pheromone on trails. The other ants coming across the trail follow the marked patch instead of wandering randomly in order to check the food source. If they become successful to find the food they will return the nest and reinforce the pheromone on the trail. In order to select a trail, each ant makes a local decision by comparing its own experience with the environmental information (trails) which is updated by other ants and finally it selects the strongest path(trail) in terms of the density of pheromone considering the fact that the pheromone evaporates over time. In other words, in the intersections the ants prefer to choose the strongest trail through comparing the already available trails marked and modified by different ants (i.e., indirect communication or stigmergy) instead of direct communication and exchanging information with other ants. Using this approach the pheromone of the paths with the long distance source will be more evaporated than the shorter paths since for the long paths it takes more time for the ant to reach the nest. Thus the density of the pheromone in the shorter paths would be higher than the longer paths which leads to discover the closest resources with higher probability. The ACO based search methods make benefits [265] for resource discovery in terms of autonomy (the nodes do not have any global information), parallel search, proximity-awareness (in terms of quick convergence to near optimal solution), efficiency (prevents a large-scale flat flooding [267]), flexibility (in terms of supporting multi-attribute range query [267]) and robustness (in terms of system work load). There are several proposals for ACO based search methods such as Semant [268], NAS(Neighboring Ant Search) [269], ACS(Ant Colony System) [270] and Max–Min Ant System [271].
Neural Search [272]	Neural search is based on Artificial Neuron Network (ANN) [273] wherein a set of neurons (computing units, nodes or computing elements) are connected together in order to construct a network which mimics a biological neural network of the brain. Artificial neurons are the simple modelings of brain cells which are connected using both of feedback and forward links with different dynamic weights that create an adaptive system. The adaptive weights are the numerical parameters that are tuned by a learning algorithm during run time (or training/prediction phase) and conceptually they clarify the strength of the links which can be used for the link evaluation in the process of neighbor selection of the neural search. NeuroSearch [274] is an example Neural search which attempts to solve the overhead problem of the flooding based approaches such as BFS by selecting the best neighbor for query forwarding based on the individual evaluation of the output of the neural network (for a set of specific input parameters) for every one of the neighbors. ANN will create a deterministic or probabilistic maps between input and out parameters which leads to specify the weights for each one of neighbor nodes.
Viral Search [275–277]	Proposes a meta-heuristic search method based on viral infection using a biological analogy. It takes the advantageous of Multi Agent Systems (MASs) [278] and combine it to some well-known approaches in Arithmetic Intelligence (AI). The viruses in the system are considered as part of the general infection, while each individual tries to make its own benefits but leading in the overall benefit of the viral system. It is desirable for viruses to infect the individuals with the minimum level of health. The efficient viral infection can be achieved by continues searching the individual microorganisms (e.g., bacteria) that are appropriate for infection (i.e., the best solutions which are the individuals with the less level of healthy). Along this way the viruses optimize their search results by infecting less healthy individuals (poor solutions). The probability to extend the domain of infection (in other words, achieving higher access to the new potential results) can be increased by systematically propagating the viruses which are lodged on unhealthy cells .

There are a lot of similarities between the behavior and characteristics of the components of any distributed systems (e.g., P2P overlay) and the behavior of bio-organisms in the biological systems such as birth (joining new member), death (node departure or failure), migration, replication, division, finding the food sources (resource discovery), chemical signaling (communication messages). As an example of resource (food) discovery in biological systems, a bacteria, is able to release a chemical signal that produces chemical gradients in the environments. Thus other bacteria in vicinity can detect the gradients and know about the location of other signal-emitter bacteria. This mechanism can be used to disseminate the

location of the source food, which has been already discovered by any one of the individual microorganisms. Another example is the mechanism that a colony of ants collaboratively use to discover the food source, and build the shortest path to the colony.

Bio-inspired search algorithms for resource discovery have been studied and proposed in several research works. As example of this kind of search methods we can mention Tabu Search [279, 280], Ant Colony Algorithm [264, 265], Immune Search [275], Neural Search [272], Viral Search and Bee Colony Algorithm [258, 262, 281, 282]. Table 2.6 briefly describes some of these approaches.

2.4.4 Packet Propagation

Unicast is the most conventional communication method in resource discovery systems. The origin node (sender) explicitly addresses the receiver and sends any type of messages (e.g., query, reply, advertisement, etc.) directly to the destination node through the network. It is effective, since the receiver address is clear and it does not require to occupy the network bandwidth with unnecessary extra communication messages. Also, the sender can know if the receiver already received the message. The problem is that in a pervasive distributed environment with dynamic resources, the destination nodes are not always known in advance, thus it is required to employ other alternative propagation methods such as broadcasting, multicasting, publish/subscribe, manycasting and anycasting.

Broadcasting or simple flooding, uses one to all communication mechanism to send packets for all the nodes in the network or subnet. Resource discovery systems employ broadcasting in the absence of predefined servers, which creates a lot of overhead in the network.

Multicasting for a number of nodes starts through establishing a multicast group by sending unicast initiate messages to each other. Once a node sends a message to a multicast address, it causes several unicast messages from the sender to all the members of the multicast group. Multicast is used when the destination nodes are unknown at the source of the message at the network or link layer and it is not clear which nodes are offering the requested resources for a particular query. This communication model, compared to broadcast, is still efficient. Publish/Subscribe [283–285] is similar to multicasting in nature; a receiver node subscribes certain services, which are offered by publisher nodes, and obtain these services directly or through intermediate nodes. Publishers constantly report all the service changes to their subscribers. They may also update intermediate nodes. Subscribers are not updated immediately and they should fetch new data from the middle nodes as soon as they contact them, instead of direct updating.

Anycasting provides the capability to communicate between a source node and one member (the one closest to the source) of the anycast group (a group of target hosts). In other words, a single anycast request will get a single reply, which leads anycast to be considered as a strong way to make reliable scalable and transparent communications, with stateless distributed services such as resource discovery and DNS. For example, in DNS, a number of replicated DNS servers create a anycast group where they are listening to a common anycast address. Upon getting a request the DNS server with the shortest path to the requester will reply the request [286]. For the purpose of resource discovery, due to the inherent single request single response (from the nearest member) feature of the anycast communication model, and its capability for coarse-grained load balancing between the members of anycast group, it is useful as a transparent resource discovery primitive.

Manycast integrates the characteristics and advantageous of multicast and anycast, leading

to an efficient communication paradigm, wherein a source node sends the packets to a certain number of pre-defined members of a multicast group. In the following of this section, we mainly focus on anycasting since we consider anycasting as a powerful paradigm for managing and locating resources in decentralized distributed systems.

Despite the aforementioned strengths, anycasting has some limitations and weaknesses [287]. The first limitation is that session based applications, such as all the implemented applications on top of the TCP layer can not benefit of the anycast addressing mode. This happens because it is possible that the subsequent packets from the same source node (and session) are routed to a different target host member of the anycast group (it has no knowledge of the TCP session state). The second problem is that anycast IP routing is inherently not scalable. In fact, the routes to different anycast groups in the routing tables cannot be aggregated. Widespread adoption causes a huge growth of IP routing tables, which reduce system scalability. If the members of a anycast group are scattered over the Internet, in each routing table, for each anycast group, it is required to have a distinct routing entry; this is costly due to the limited efficient size of the routing tables. Moreover, the dynamic behavior of anycast members (joining and leaving members) leads to frequent changes of routing configurations (i.e., frequent changes in network topology), which possibly makes the system unstable, so that the routing and discovering protocols can not operate efficiently. There is another problem with anycasting that might affect its performance. The problem is due to the inherent static target member selection of anycasting which is based on the fixed indicator of the shortest path. In fact, it doesn't support multiple target selection constraints such as network congestion and current target load. Due to the above strengths and weaknesses, TCP based services are not able to take advantage of anycasting.

However, anycasting, with its native robust capability to efficiently find nearby resources has been considered as an important communication paradigm to enhance the resource discovery approaches by leveraging the aforementioned features and capabilities through implementing an anycast based system. PIAS [181] and ASTAS [182] are two examples of anycast architecture which can be used to implement anycast based resource discovery systems.

PIAS is an IP anycast architecture, which makes use of a proxy overlay for advertising IP anycast addresses on behalf of group members and tunnels anycast packets to those members [181]. ASTAS is an architecture for scalable and transparent anycast services, which is based on PIAS. It establishes an overlay infrastructure compound by two types of nodes: Client Proxys (CPs) and Server Proxys (SPs). Both of these proxy nodes act as routers that advertise routes from their neighbor nodes to a anycast IP range into the routing substrate, by forcing IP packets containing the anycast members as the destination address to pass through the overlay. Once a client node initiates a new session towards an anycast destination, the nearest CP registers the new session, and afterwards it selects the most suitable SP forwarding the request to it. Upon receiving a new session request by an SP, it selects the most appropriate server node to process the request. Because of the stateful nature of proxy components, ASTAS provides more fine-grained and load balanced distribution of the service request over the available resources than PIAS.

2.4.5 Information Delivery

Individual nodes of a resource discovery system need to share their resource information in an efficient manner. Thus the information delivery mechanisms must aim at generating little network overhead and bandwidth consumption, through the reduction of the volume and

size of exchanged messages. To achieve this purpose there are some well-known techniques such as caching, piggybacking and heartbeat checking.

Caching relies on storing successive resource advertisements of each node, so that new queries for similar resources can be resolved locally. For example, when a resource matching a query is found in a node, that node will send the resource information backwards to the original requester, which leads that the requester node and each of the intermediate nodes can cache the information. For the new queries, each node first checks its local cache to find the query match and, if local matchmaking is not feasible, the query will be forwarded to other nodes in the system. The system has to manage stale-information stored in the cache by either removing expired cache entries or by eliminating the oldest cache entries, in order to add new entries when the cache size limits are exceeded. Using caching mechanism can improve system efficiency (in terms of overhead, latency, bandwidth consumption and query work loads) by handling the queries locally instead of doing global query processing. Since consequent computing applications or application segments frequently require almost similar resources, caching the result of the previous queries might be useful to avoid repeating the same queries in the next discovery cycles.

Caching-based techniques have been widely deployed in many resource discovery research efforts. For example, some works [202, 257] propose an efficient resource discovery protocol based on proactive information caching, which supports the construction of self-structured overlays. They have an ant-inspired algorithm that uses selective flooding to exploit local caches, in order to decrease the number of explored nodes for each query. Caching mechanisms have been further improved through periodic exchanging of the cache contents between neighboring nodes, through an epidemic replication mechanism based on gossiping. Leveraging the caching mechanism enhances the system performance by reducing the number of hops to locate required resources and decreasing the overall discovery loads.

Piggybacking means responses can be included on top of acknowledgment messages. In other words, a regular reply message can carry (piggyback) extra information. Piggybacking might be costly in terms of the amount of information transacted in the system. However, it can significantly reduce the number of messages to resolve a query. For example, works include [288–291] use piggybacking technique to facilitate information delivery. Unlike piggybacking, in heartbeat checking [292], a sender node periodically sends a signal (a very small amount of information) to other nodes, updating the status of the sender (e.g., in terms of availability). This mechanism is more compatible with the Proactive discovery strategy, where a resource provider periodically advertises its resource information to the clients (e.g., subscribers).

2.4.6 Query Termination

In general, the resource discovery process starts by sending a query message to the network, where intermediate nodes/agents spread the query through different search and forwarding techniques (See Section 2.4.3). However, the query must be terminated at some point under some specific conditions, otherwise the discovery procedure would not be under-control, resulting in the creation of extra unnecessary network overhead, without having a guarantee of finding the required resource. The following common situations describe how a query can be terminated, either by natural mechanisms or by a query termination method.

A) The resource required for the query is found in the current node, in which the query instance resides, thus the query is successful and the query response must be sent to the original requester either through direct communication or through back-tracing.

B) The query already has visited all nodes in the system, therefore the query is stopped and deleted. The last visited node may report the query status to the originating node. The results inform the originating node that the required resource does not exist in the system. Queries leave marks at each explored node, indicating that this node was already visited by the query. However, using blind searches (for example, in an unstructured system) it is a challenging issue to ensure that all nodes in the system have been explored by a particular query, since, the number of the nodes in the system as well as the network topology is unknown. On the other hand, even in a known system, a query instance may know how many and which nodes were visited by itself. But it would be more complex to know about other instances of the query, since communication and synchronization between several instances of the same query generates extra overhead.

C) The query is not able to be further forwarded to the next hop. Thus the query is stopped and deleted. This happens because of two possible reasons: first, the current node is a leaf node, which means that there is no more neighboring nodes to forward the query along, or the query was already sent to all neighbors. The second reason is that the query's TTL has expired.

Iterative Deepening is one of the termination techniques that can be used when the TTL expires while the query is not resolved, and perhaps the search space was not completely explored. If the originating node, after finishing an iteration (i.e., upon receiving the TTL expiration notifications from all the instances of the query) does not receive a query response regarding its required resource, then it starts a new iteration by resending the same query with increased TTL, to search within larger radius of the search space. This process is repeated in the next iterations until either the query is resolved or the query's specific timeout is reached. The query timeout may be different from TTL, which generally is the number of visited hops by a query. The drawback of this solution is that, in each iteration, the query propagates through a substantially larger portion of the same nodes that have already been visited in the previous cycles, creating unnecessary search overhead.

Checking [233] is another termination mechanism which can solve the problems of Iterative Deepening. When a query-replica reaches a point that its TTL is expired, the node sends a specific checking packet to the original requester. Upon receiving the checking packet, the requester will see if the query already is resolved by other query instances. Otherwise it sends back the checking packet including the information to increase the TTL of the query replica and then the query proceeds being forwarded. Generally, each query replica sends the checking packet either periodically after n hops or after TTL expiration. The drawback is the use of extra communication to exchange the checking packets.

Chasing Wave [293] is another possible approach where upon receiving a query response from any one of the query replicas, the originating node sends chase packets to chase and stop the other active query replicas. This mechanism works based on the concept that chasing packets traverse faster than query packets. This can be done by increasing the delay before forwarding as the distance of the query replica to the originating node becomes larger. The paths to the active query replicas can be traced using their marks on the intermediate nodes, which indicate each query's next hop. Using this method may impose slower discovery process by increasing the delay before a query is forwarded at far nodes.

Backward Synchronization is another alternative termination approach triggered when a query-replica faces a deadlock and the query is still not resolved. The deadlock might happen, when there is no neighboring node (except the node where the query came from) to forward the query or if all neighbors already have been visited by other query-replicas, or in the case

that the query’s TTL is expired and we do not want to extend the search diameter to discover far distance resources. In such a situation, the current node, sends out the query backwards to its parent node (the previous node that sent the query replica) while it decreases the TTL of the backwarded query. The backward query message will proceed to the parent node in the normal way (for example, by forwarding the query to a randomly chosen neighboring node who has not been visited by other query replicas). In any stage, if the query faces a new deadlock, the backwarding is replicated unless the query is resolved or the originating node becomes the parent node. Like the Chasing Wave approach, queries leave marks on visited nodes indicating their next hop. This information is used for backwardness and determining the visited neighbors by other query replicas. This Backward Synchronization method can be used for search methods like Random Walk. Using this technique along with flooding approaches is not efficient due to the fact that, in flooding based search methods, all the possible behind paths already have been explored leveraging a large enough number of query replicas.

2.5 Evaluation Aspects

Table 2.7 provides an overview of the most important performance factors (i.e., evaluation metrics) to assess resource discovery protocols for distributed computing systems [294–297]. In the remaining of this section, we further discuss the evaluation strategies to measure these performance factors.

Table 2.7: Overview of the evaluation methods for resource discovery.

PF	Evaluation Methods	Description
Scalability	Quantitative Assessment (e.g., Overhead, Discovery Latency, Bandwidth Consumption, Query Load)	The capability of the resource discovery protocol to maintain its performance regardless of the other system parameters such as network size, query dimension, and environment dynamicity. For example, in a large many core system, we must clarify whether the discovery procedure is system size independent in the case of performance, throughput, and overload or not. Besides, the discovery protocol must scale up and scale out efficiently across multiple / many cores, multiple / many processors.
Efficiency	Quantitative Assessment (e.g., Network Overhead, Discovery Latency, Bandwidth Consumption, System Utilization)	The ability of the system to be cost efficient in terms of computation and communication cost in all of three discovery stages: establishing the system, performing resource discovery and system maintenance. (e.g., the discovery must be fast as much as possible). The ability of the system to maximize the utilization of all the potential resources (e.g., idle cycles).
Reliability	Quantitative Assessment (e.g, Rate of successful/un-successful querying, Recovery Time)	They ability of the system to prevent and manage faults. (e.g., unavailability of peers/resources) The discovery results must be accurate. (e.g., nondeterministic search results)
Flexibility	Qualitative Assessment	The ability of the system to perform different type of Complex querying such as Multi-Dimensional, Range, Multi-Dimensional Range, Aggregate, Nearest Neighbor , Exact, Partial and Keyword Querying.
Dynamicity	Qualitative Assessment	The ability of the system to cover different kinds of dynamicity. (e.g., dynamic attribute values of resources, frequent arrival, departure and failure of the nodes)
Heterogeneity	Qualitative Assessment	The ability of the system to work in the environment with different type of heterogeneous resources. (e.g., Low-Heterogeneity: resources with different attribute values, High-Heterogeneity: resources with different set of attributes)
Clustering	Qualitative Assessment	The ability of the system to establish a self-organized overlay or platform which can support some valuable inherent features such as proximity-awareness, semantic-awareness and QoS-awareness
Autonomy	Qualitative Assessment	The ability of the system to be purely decentralized. The autonomy of the peers to local control and manage their data and behavior.

2.5.1 Efficiency

The efficiency of a resource discovery solution can mostly be represented and demonstrated along the following conceptual metrics and functionalities.

Discovery Latency: One of the most important performance metrics for resource discovery protocols is the discovery response time or the end-user latency. The discovery latency in distributed systems might come from different sources, which are mostly related to the network transfer times, query processing algorithms, type of querying (e.g., number of dimensions in multi-dimensional querying), network size, computing environment and the amount of parallelism (e.g., the number of walkers). In the case of network transfer times, the potential efforts to minimize latency is to reduce the number of bytes sent and also the number of times they are sent. Thus, the discovery mechanisms and algorithms are required to be optimized to transact the discovery messages (e.g., requests, replies, updates, etc.) with regards to minimizing network bandwidth. Along this line, metrics such as discovery load (i.e., the average number of transacted messages per query), number of hops per query, and the number of explored (visited) nodes/resources per query can be used.

Load Balance: During a discovery procedure, the query/discovery load will be distributed among all potential resource information providers. Employing a load balanced technique for querying will enhance the total scalability of the system. Furthermore, since resource information providers are completely distributed in the system, and querying mechanism follows a symmetrical distribution, it also prevents any kind of centralization, bottlenecks and single points of failures.

2.5.2 Scalability

Resource discovery scalability represents the ability to cope with variations in the system effectiveness, where the effectiveness is measured in terms of the system's performance, throughput and QoS seen by the resource discovery clients (e.g., end users and job schedulers in Grid or Cloud, process managers in distributed operating systems, applications in HPC), while a set of critical system parameters are changing (e.g., the system size is increasing) (see Figure 2.18).

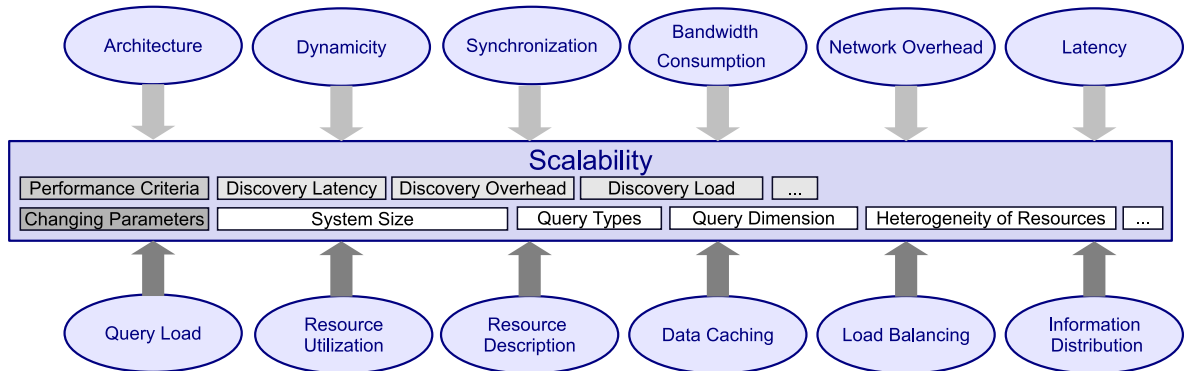


Figure 2.18: The important elements which have impact on the scalability of a resource discovery system.

It must be taken into account that scalability is one of the most important key factors

to evaluate distributed systems in large scale environments. Referring to the nature of large scale computing in LDCE, system size is the most significant changing factor for scalability evaluation in the majority of the resource discovery research works (other changing parameters, such as query types and query dimension, are occasionally considered for study). Increasing the system size has impact on the system performance by creating larger communication overhead, which increases network latency and results in an increase of the discovery response time (or discovery latency). Thus, the communication overhead can be measured in terms of number of transacted messages among the distributed nodes during query processing for a particular application resource request.

The ability of the system to grow in terms of available resources/peers, such as cores/nodes, without impacting overall system performance will impose conditions/restrictions in various aspects such as system architecture, overlay construction, communication model and query processing. In the remaining of this section, we address these aspects.

In order to avoid any central point of failures and bottlenecks, discovery systems ideally should avoid relying on a centralized architectures. However, implementing an efficient fully distributed design is not attainable or has drawbacks as well. Therefore, depending on the case, an hybrid architecture such as a distributed hierarchy might be useful. We must notice that on a distributed system it's impractical to have a global view of the system while making it scalable, instead it is possible to make specific queries to the system but it leads to a more limited view of the system. Furthermore, the distributed architecture must be loosely coupled between nodes. The advantages is that nodes are not affected by failures in other nodes, and failures become very rare and limited to the point of failure, only affecting a limited number of clients (e.g., applications) that use resources in the point of failure.

The ideal system behavior would be for the communication overhead (number of exchanged messages) or query response time (as the result of decreasing overhead and latency) to stay constant as the number of nodes/resources (e.g., processors) increase. In a system that is not scalable we will expect that the system eventually reaches a point where the discovery latency/overhead grows significantly worse with the system size. Due to the aforementioned impacts on scalability of the other important system factors (such as performance and resource utilization) it is possible to evaluate the scalability of the discovery system by studying and analyzing the system effectiveness.

In order to create a highly scalable system, discovery systems should support dynamic behavior and dynamic characteristics of resources, while tolerating failures and support load balance. Furthermore, the components of the discovery system should communicate asynchronously and efficiently, i.e., avoid high throughput in communication. Asynchronous querying gives freedom and autonomy to the peers, or enables discovery components to have an independent control over their data and behavior regardless of the global system structure, which increase the level of system decentralization. In other words we can measure the system scalability based on the extent of its distance from a fully centralized system.

When system size increases, resource discovery must enhance resource utilization by providing support to make an efficient use of the increasing available resources, while avoiding overhead. It must perform load balancing based on resource demands while it avoids excessive communication overhead in terms of latency, query workload per each node, memory access, synchronization and also traffic congestion in the communication routes within the system. We must also take into account that some amount of synchronization and message transaction are potentially required to maintain the discovery system (e.g., recovering from failure, handling departure of the nodes or arrival of new nodes)

The problem of scalability comes from different sources. For example, communication between the large number of resources will increase communication overhead, latency, jitter and causes packet loss, which results in a scalability problem. Some other significant sources are memory accesses, synchronization and signaling. Frequent remote memory accesses, because of data dependencies or memory constrains, increase memory access latency. Data or state synchronization generates communication traffic and finally signaling overhead of communication protocols (state maintenance) has scalability concerns. In a scalable system, it is expected that the system performance (e.g., in aspects like discovery latency or overhead regarding storage, communication and computational load) is independent from system size and the resource discovery scales with increased system resources. We can evaluate the scalability of the discovery system by assessing the impact of changing the parameters of the computing environment such as network size in the aforementioned scalability issues. Table 2.8 provides some examples of the evaluation of scalability methods in current resource discovery literature.

Table 2.8: Some examples of the evaluation of scalability methods in other resource discovery research works (RD: Resource Discovery, ET: Evaluation Tools, PL: Planet Lab, BM: Berkeley Millennium, IF: Iamnitich & Foster, SG: StarGrid, MT: MatchTree, OS: OntoSum, NA: Not Available).

RD	Measured Criterias	Changing Parameters	Scalability Evaluation Results	ET
Ganglia [51]	Performance overheads , average bandwidth consumption per node for monitoring in a single cluster , total bandwidth for aggregating data from a set of clusters	The number of nodes within a cluster and the number of sites being federated	Linear scaling in packet rates, Linear scaling in local-area, Bandwidth consumed as a function of cluster size	BM, PL
OS [192]	The metric of recall rate, which is defined as the number of results returned divided by the number of results actually available in the network	Number of nodes in the network	OntoSum's recall decreases less with the increase in network size	NA
MT [62]	Query response time, Bandwidth consumption, Query completeness	Query Types (easy and difficult queries)	A hierarchical result aggregation provides balanced processing overheads among resources with scalability. Several heuristics are proposed to improve the query response time and to decrease overall network overheads, and they are evaluated through large scale simulations. It doesn't present any direct evaluation method to evaluate the scalability	PL , OverSim
SG [298]	Throughput and query response time, Throughput is the average number of queries processed by the resource discovery component per second	The size of the Grid, number of nodes, number of concurrent requesters	The results prove the scalability for the limited size Grids	NA
IF [299]	Response time as the number of hops traversed for answering a request, success rate, which is the percentage of successful queries considering the dropped requests because of dead ends or exceeded TTL	The number and the frequency of shared resources/nodes, Different values of TTL, Under different sharing and usage policies (different number of existed resources for each particular resource type)	Evaluation on the top of the proposed resource discovery architecture shows that the relative performance of the different forwarding algorithms considered is independent of the average resource frequency. The resource discovery performance is correlated to the sharing characteristics.	Testbed

2.5.3 Reliability

Reliability is one of the major concern for resource discovery particularly in a HPC environment, which requires reliable fast execution of the tasks on computing resources. We

can distinguish between different types of reliability for resource discovery with respect to the following aspects:

Accuracy (also known as Correctness): It concerns the accurate description and representation of queries and resources. In other words, accuracy is how efficiently the query/resource description model of the resource discovery protocol represents the features and complexities of the real world applications and hardware. For example, a simple flat attribute-based resource description model is not able to precisely demonstrate the characteristics and behavior of the hardware processing components such as memory hierarchy, many-core architecture and interconnection networks in a large scale HPC environment. Thus, the resulting resource discovery approach might not be reliable in such environment. Additionally, in other example, providing a query description model that only supports exact match, single resource querying, would not be able to satisfy the resource requirements of a parallel multi threaded application, which simultaneously needs to discover a group of connected heterogeneous resources with particular QoS communication values along each connected edge. In fact, the accuracy evaluation of the query/resource description model is part of the resource discovery evaluation, which partially specifies the reliability of the discovery.

Reliability also concerns a precise result of resource discovery procedure (i.e., returning results that do actually meet the requirements of the resource requester). It means that the discovered resources must carefully satisfy all of the query conditions. Along this line, the success rate and hit rate are two important criterias to evaluate the reliability of the resource discovery approaches, while these evaluation factors as well as cost per query and cost per hit can also be conducted to evaluate the system efficiency [257, 300–305]. A query is considered to be successful when at least one existing resource, matching the query, is detected. We note that a resource discovery solution is known to be deterministic if any query with an empty result can assure that even one resource matching that query is not existing in the whole system. Thus we can conclude that a deterministic resource discovery process provides reliability at least in terms of accuracy and success rate. Similarly, the hit rate (i.e., recall rate, success rate) measures the percentage of successfully discovered resources out of all matching ones (see Equation 2.1).

$$\text{Hit Rate} = 100 \times \frac{\# \text{Hits or } \# \text{Discovered Resources}}{\# \text{Matching Resources}} \quad (2.1)$$

Dynamicity and Fault Tolerance: Reliability must cover the dynamicity issues. The resource dynamicity means that any node/resource in the system can join, leave or fail at any time (i.e., dynamicity in terms of topology change as well as resource life-cycle) and also it can change its characteristics (i.e., dynamicity in resource conditions and availability) [306, 307]. Thus, resource discovery performance must be evaluated using different configurations and dynamicity parameters (such as churn rate which is the percentage of nodes/resources in a given time frame that fail, leave or join the system) for the environment. In fact, in an unreliable system, it might be possible for a requester to get a resource discovery result containing information about a resource/node that already has left the network.

Resource discovery solutions must have mechanisms to deal with failures. These come from different sources that are mostly related to centralized discovery architectures (i.e., single point of failure and bottlenecks), dynamicity issues (i.e., dynamic attributes, node/resource joins, leaves and failures) and churn conditions. Tolerance to failures and churn is expected to be a basic characteristic of discovery systems particularly in very high dynamic environments containing large number of non-dedicated resources. Similar to the resource discovery evaluation

in terms of supporting dynamicity, the discovery system must be evaluated for different configurations range from very stable to very volatile environments to prove the system toleration in different levels of churn that are to be expected from non-dedicated resources [308, 309]. There are several research works in the current literature [310–312], which analyse the impact of churn on the discovery performance. As examples of these works, the results in [313, 314] conclude that the DHT based discovery solutions cannot efficiently cope with high churn rates and they probably generate some non-tolerated failures like unexpected time-outs (caused by a finger pointing to a departed node) or lookup failures (caused by nodes that temporarily point to the wrong successor during churn) [315]. ECHO is a recent work which copes with DHT churn problems through implementing a tree-based index structure on top of DHT overlays [56].

Resource Reservation and QoS: Resource reservation capability is required for resource providers to reserve resources for particular applications (like multimedia applications). This means that resources are only freed when their associated processes are done. In other words, resource reservation ensures that the resources required are available for the processes upon the need.

Validity: Each resource can have an expiration time where after a specific period of time, the resource becomes unavailable. It might happen that the discovery results contain an invalid resource which has already been expired.

2.5.4 Flexibility

Flexibility refers to the ability of the discovery system to express the capacity of resources and query requirements, both provided and sought, in a form that is convenient for the resource discovery users. The discovery mechanism must allow querying with very specific and detailed conditions (i.e., complex querying) as well as loose ones (general querying). Here, we elaborate the most important querying functionalities, which represent various aspect of flexibility for resource discovery solutions. Employing these significant querying features enriches the resource discovery module to be more powerful and applicable to fulfill the requirements of various applications, configurations and computing environments.

Multi-Dimensional and Range Querying is the capability of the discovery system to process queries containing multiple attributes either dynamic or static in specific ranges of values. The discovery results must satisfy every one of the conditions for each particular attribute. This would be more complicated specially when each single attribute is required to be qualified within a particular range of values. Since the resource contentions along multiple dimensions might happen in the environments with the uncoordinated analogous queries, the problem of multi-dimensional range-querying is known to be a challenging issue. In addition, the multidimensional querying will probably reduce the rate of resource matching, while increasing query latency and bandwidth consumption.

The impact of multi-dimensional range-querying can be evaluated by using evaluation factors such as Failed Task Ratio (F-Ratio) and Throughput Ratio (T-Ratio) [316] which directly reflect the rate of resource matching for any resource discovery protocol. F-Ratio is the ratio of the of the number of failed (unsuccessful) tasks (i.e., the tasks that cannot discover any qualified resources) to the total number of generated tasks in a particular time period. T-Ratio also refers to the ratio of the number of completed (executed) tasks to the total number of generated tasks within a specific time period.

In the current literature, there are several resource discovery solutions that support multi-dimensional, range and multi-dimensional-range querying through leveraging several techniques. As examples of these approaches we can mention CAN-based protocols (e.g., CAN [72,317,318], RT-CAN [319], PID-CAN [320]), MAAN [321], Mercury [77,322], Armada [323], Murk [324], Skip-Tree [325], SWORD [326], Node-Wiz [327] and MatchTree [62]. Moreover, a comprehensive survey on multi-dimensional methods and techniques can be found in [328].

Resource Graph Discovery is the capability to discover a graph of resources considering both individual characteristics and interconnecting properties of resources. For example, in order to allocate resources for the threads in an application graph, we must consider the communication conditions and limitations among the application segments, therefore, the classical approaches of resource discovery, which are mostly relied on finding resources according to the individual resource characteristics cannot be efficient. In fact, resource graph discovery is the process to find and select the best possible matched graph of resources in the system where all the edges and vertexes can satisfy the query conditions.

Aggregate Querying generally relies on tree-structures to aggregate and combine results from a large number of nodes in hierarchy. The common examples of aggregation queries are Average, Median, Count, Sum, Maximum, Minimum and Top-K [329–333].

Nearest Neighbor Querying (or k Nearest Neighbor Query - k NNQ) means that for each given query, the discovery system must be able to return the k -Array = $\{r_1, r_2, \dots, r_k\}$ of results, in which r_i is the i -th nearest qualified resource of the requester. In other words, the resource discovery provides the list of discovered resources considering the priority of the closer neighbors.

Exact Matching is one of the general search functionalities which can be supported in most of the resource discovery systems. However, structured overlays such as distributed hash tables, provide better support for exact matching, in comparison to the other systems, through forwarding the query to the node which values of its keys precisely match the query's requirements. In this kind of querying, typically, the query looking for "the resource whose key is K ", due to the overlay mechanisms will be forwarded to the peer which is responsible for the address $\text{hash}(K)$. Thus, the querying operation results in a forwarding operation and its efficiency would be directly dependent to the forwarding efficiency on the overlay.

Partial Matching and Keyword Search, are challenging for structured discovery systems. In many querying cases, for a given exact matching query, the results might be empty while in the system, there still exist resources that can partially meet the query conditions. i.e., for most applications, even the partially matched results (depending on their matching degrees) can be acceptable and provide benefits to requesters.

In keyword searching, users ask for any resource containing a given set of keywords advertised in (or extracted from) its meta-data or its resource description. Structured overlays such as pure DHT based discovery systems (e.g., CAN, Chord and Pastry) have addressed several of the issues related to the reliability and scalability that plagued the traditional P2P overlays (e.g., Napster and Gnutella). However, the advantageous functionalities such as keyword searching and partial matching presented in Napster and Gnutella are missing in DHTs that aim to replace them [334]. On the contrary, in unstructured systems like GIA [335], these types of querying are easily supported, basically because indices are mostly local and for

a query visiting a peer, it would be easy to lookup the index table for a resource having a set of required keywords.

In-order to support keyword-search in structured systems, generally, there are some typical approaches such as Inverted Indices [307,336] and Bloom Filters [303]. The first method consist in the creation of inverted indices where, for each keyword k , the inverted index maintains the pageranks. The pagerank value specifies the priority of a resource for a particular keyword with regards to other resources, i.e., it is used to order indices at any resource. In addition, the inverted index stores the pointers to all of the resources/nodes in the system which announce that keyword in their descriptions. Thus, a potential approach is to record the inverted index for the keyword k at the node which is responsible for address $\text{hash}(k)$. The basic mechanisms to answer a keyword query containing a set of keywords $K = \{k_1, r_2, \dots, k_n\}$, in most of the optimized DHT-based discovery systems [337] such as Kademlia [338–340], includes two steps, where in the first step all inverted indices relative to each keyword k_i , $\{i = 1, \dots, n\}$ will be retrieved, and in the last step these indices will be intersected to achieve a set of resources that contain all the keywords in their resource description or meta-data. The inverted indices guarantee to provide optimal hit rate, but they potentially can generate a huge amount of network overhead particularly when the large number of resources present a required keyword in common [341]. This happens because the inverted indices retrieved have to be sent to a single node in the network, which is responsible to perform intersection.

Bloom Filters (BFs) [20, 234, 342, 343] are another solution to perform keyword searching in structured systems. A bloom filter can be defined as a hash-based data structure which summarizes membership in a set. Thus, for the purpose of intersection between the keyword sets of the individual resources/peers in the system, it is possible to transfer a BF of a set instead of sending the set itself. This significantly reduces the amount of communication required for supporting keyword searching in the system.

Other than Inverted Indices and Bloom Filter, there are some other well-known supportive techniques such as replication, partitioning, caching, ranking and incremental results [334], which facilitate performing the keyword searching on top of structured overlays. Several works [303, 312] provide surveys on the various aspects of the search methods containing keyword lookup, range queries, multi-attribute queries and aggregation queries.

2.5.5 Heterogeneity

The heterogeneity of a resource discovery solution can be evaluated along the following different aspects:

Various Administration Domains: The peers and resources can potentially belong to different underlying administrative domains and sub domains, particularly in large scale, geographically distributed computing environments (e.g., Grid). Evaluation of resource discovery approaches on top of the real test-bed environments such as PlanetLab and OneLab might be a proper option to evaluate heterogeneity in terms of various administration domains. PlanetLab is a global research network consisting hundreds of nodes, geographically distributed in various sites belonging to different academic institutions and industrial research labs.

Various Hardware, Software and Infrastructures: The resource discovery approach can be adapted to work simultaneously in different computing environments with various types of resources in terms of hardware (i.e., various hardware infrastructures) and software platforms (i.e., heterogeneity of the runtime environments, network protocols, operating systems, different data representations, data models, data structures and data accessing

policies). In fact, resource discovery can be evaluated in presence of heterogeneous computing cores (i.e., heterogeneity of the resources in a cluster or even inside the same system and chip, multi-core and many-core processors, CPU, GPU, Field Programmable Gate Array (FPGA), Cell Broadband Engine (Cell BE), GPUs mixed with CPUs), heterogeneous hardware implementations (i.e., various architectures and memory hierarchies), heterogeneous Network on Chip (NoC) latencies (i.e., various networks and interconnections, heterogeneity in the Internet). As an example, the resource discovery is required to find heterogeneous resources for the purpose of code adaptation in HPC and cluster computing through adapting the machine code on the fly to different hardware platforms and operating systems, which is one of the most difficult scenarios considering heterogeneity.

Various Applications: A resource discovery approach can be created for different purposes. Thus, due to the resource discovery objectives, it can be used by different type of users, system components or applications. For example, a resource discovery protocol which is designed for the purpose of distributing task execution and task migration in computing environments such as Grid, Cloud or HPC must provide the requirements for heterogeneous applications such as HPC and Real-Time applications. HPC applications have to make use of the vast amount of parallel resources. This requires that the HPC application itself can exploit as much as concurrency as possible. In other words, HPC applications are highly concerned with scalability constraints to support high degrees of concurrency [344]. But, Real-Time applications are more concerned with timing issues. They are not aiming to increase the level of parallelization. Rather, they are specifically interested for handling their own timing constraints which bears resemblance to the synchronization estimation (i.e., execution timing) in distributed applications. In other words, by exploiting the timing constraints applicable in real-time scenarios, the distributed synchronization behavior can be improved further.

2.6 Comparison

As we discussed in previous sections, there are set of important functionalities, features performance indicators for resource discovery solutions in order to meet all the major requirements of different applications in various computing environments. Along this line, we have elaborated the most significant resource discovery evaluation factors and criterias in terms of Scalability, Efficiency, Reliability, Flexibility and Heterogeneity. In this section, we select three major groups (in terms of popularity) of resource discovery solutions for distributed computing environments: Grid-based (e.g., MDS-4, OntoSum), P2P-based (e.g., SWORD, MatchTree) and Hybrid-Approaches (e.g., NodeWiz, CycloidGrid).

Moreover, we choose some of the well-known approaches as representative of each group and we compare them based on the evaluation factors that are mentioned in the previous sections. Table 2.9, 2.10 and 2.11 provide a summary of comparison between the aforementioned resource discovery solutions considering the qualitative assessment through different evaluation factors.

As shown in Table 2.9, Grid and P2P based approaches (like OntoSum and MatchTree) provide better scalability compared to Hybrid approaches. From Table 2.10, we see that, P2P based approaches (SWORD and MachTree) are the most flexible resource discovery solutions among others. But their capability to deal with the heterogeneity (of resources) is lower than that of Grid and Hybrid approaches. Furthermore, in Table 2.11, it can be seen that, P2P and Hybrid based approaches enable more sophisticated features and functionalities compared to Grid based solutions.

Table 2.9: Summary of comparison between some of the well-known resource discovery approaches for different evaluation factors.

Approach	Resource Description	Efficiency	Scalability
SWORD [326, 345–347]	Native SWORD XML syntax or a Condor ClassAd.	The load balancing mechanism in SWORD is less efficient in an environment with a non-uniform distribution of peer ranges. SWORD is a DHT-base approach and it uses multiple over lays.	++
Node-Wiz [327, 348]	It uses a relational model and standard SQL query processing in Node-Wiz-R.	Load balanced approach. Supports clustering and self-organization of the overlay. Uses single distributed indexing mechanism.	++
MDS-4 [49, 50]	It uses an extensible resource specification language based on Web Services Resource Framework.	Uses LDAP to create the overlay. It's overlay is based on GIIS and GRIS.	++
MatchTree [62]	Leverages Condor's condor-startd daemon which provides summarized system monitoring metrics while it provides resource information in a ClassAd format for matchmaking. It is developed on top of Brunet P2P overlay.	Reduces query response times through redundant query topologies, dynamic timeout policies, and sub-region queries. Leverages a self-organizing recursive-partitioning multicast tree for query distribution and result aggregation. Balanced processing overheads among resources.	+++
CycloidGrid [64]	Each resource in the system is described by a set of attributes. These attributes are CPU speed, the amount of RAM, available hard disk size, operating system, and processor model. The classification of resource attributes in this architecture is done by a decision tree (DT).	Improves the response time of user's requests. Distributes the load between peers based on QoS constraints of requests, round trip time (RTT), and current load of resources.	++
OntoSum [192]	Organizes a Grid network by a semantically linked overlay representing the semantic relationships between Grid participants. Proposes a lightweight indexing summarization scheme based on Triple Filter which is an extension of the classical Bloom Filter data structure.	Uses a semantics-aware topology construction method which greatly improves the discovery efficiency in Grids through propagating the discovery requests only between semantically related nodes. Provides the ability to locate semantically related results. In OntoSum, Grid nodes automatically organize themselves based on their semantic properties to form a semantically-linked overlay network.	+++

Table 2.10: Summary of comparison between some of the well-known resource discovery approaches for different evaluation factors.

Approach	Reliability	Flexibility	Heterogeneity
SWORD [326, 345–347]	Dynamicity in terms of Dynamic Attributes and Dynamic Topology is supported.	Supported in terms of Range Query, Multi-Dimensional Querying (using multi-query approach), Aggregate Query(Resource Graph Discovery) and Nearest Neighbor Querying	++
Node-Wiz [327, 348]	Dynamicity in terms of Dynamic Attributes and Dynamic Topology is supported. Quality of Service is supported.	Supported in terms of Range Query and Multi-Dimensional Querying	++
MDS-4 [49, 50]	Resource Reservation is supported. Fault Tolerance is supported. Dynamicity in terms of Dynamic Attributes and Dynamic Topology is supported. Quality of Service is supported.	Not Supported	++
MatchTree [62]	Guarantees query completeness. Fault Tolerance for the failures such as node churn, internal node failure and consecutive packet drops is supported. Dynamicity in term of adding new attribute is supported. Serves a different quality-of-service to users with distinct demands.	Supports complex querying including multi-attribute matching, range queries, string and regular expression matching	+
CycloidGrid [64]	Quality of Service is supported.	Not Supported	+++
OntoSum [192]	Supports accuracy in term of hit rate (or recall rate). Supports dynamicity in terms of probability to issue a query, probability to leave the system and the probability of new nodes with new resources joining the system.	Supported in term of neighbor discovery query	++

Table 2.11: Comparison between some of the well-known resource discovery approaches for different evaluation factors.

Functionalities	SWORD	Node-Wiz	MDS-4	MatchTree	CycloidGrid	OntoSum
Load Balance		✓		✓	✓	
Fault Tolerance		✓	✓	✓		
Self-Organization	✓	✓		✓		✓
Range Query	✓	✓		✓		
Multi-Dimensional Query	✓	✓		✓		
Graph Discovery	✓					
Nearest Neighbor Query	✓					✓
Resource Reservation			✓		✓	
Semantic-Awareness						✓
Proximity-Awareness					✓	

2.7 Resource Management for Large Scale Systems

In this section, we investigate resource management issues for petascale computing environments such as HPC, Grid and Cloud in the matter of future manycore architectures.

2.7.1 High Performance Computing Systems

High Performance Computing (HPC) is generally recognized as a role model for manycore systems due to the essential similarity of their architectures and hence the means of program development for both of them (HPC and manycore systems). At the same time, the manycore movement leads to a sudden boost of scale in high performance computing without essentially increasing the requirements or costs. Many HPC cluster systems employ a centralized resource allocation and management, where the jobs are being controlled centrally and statically by a single resource manager that has complete knowledge of the system. In fact, HPC clusters are generally based on a batch scheduler and resource management system which allows users to submit their jobs to a batch system where a central scheduler and resource manager entity that is hosted on the cluster-head makes decision (in terms of timing aspects and the physical location of resources) for the resource allocation to run a given job. A centralized resource manager generally manages a pool of resources and runs the jobs based on a prioritization policy. However, it may distribute the jobs among a set of other resource manager entities in a hierarchical fashion. The resources might be organized in single or multiple queues (aka partitions) based on some common characteristics of resources. Slurm [349, 350], TORQUE [128], LSF [351], PBS [352], Load Leveler [353], CCS [354] and Univa's Grid Engine (formerly Sun Grid Engine) [355] are some of the popular centralized schedulers and resource managers for the HPC clusters. HTCondor [356, 357] is another example of centralized resource management in High Throughput Computing (HTC) environments which is able to utilize large heterogeneous clusters where large numbers of relatively small-sized and short-live jobs are processed.

The future manycore enabled HPC clusters can not rely on the centralized and static resource management approaches. Resource management for large-scale manycore systems is inherently a NP-complete problem. The large number of heterogeneous cores and applications with various requirements causes scalability issues for centrally acting heuristics, where a centralized component must maintain a global view of the entire system. Resource management itself can become a bottleneck due to high computational requests and communication latencies. Furthermore, the amount of the detailed information on the resource characteristics which is maintained by the central resource managers would be limited to a very abstract level and this leads to reducing the accuracy of resource mapping and allocation (i.e., the quality of resource mapping) for the applications. On the other hand, managing dynamic resources (when resources join, leave or fail) with dynamic attributes and behaviors becomes more complex for the central resource managers since the solutions such as periodical or triggered updating, for the large number of resources and state changes, impose significant communication overheads on the system.

In decentralized resource management, multiple resource management entities cooperate to keep the workload for all computing resources in different pools or clusters balanced and satisfy the user requirements. REXEC [358], Tycoon [359] and Cluster-on-demand [360] are some examples of decentralized resource managers for HPC clusters.

2.7.2 Grid Computing Systems

In Grid computing where multiple geographically distributed and heterogeneous clusters are combined into a single system, according to the type of Grid, there exists different types of resource management system particularly designed for computing Grids such as Nimrod/G [39, 361], 2K [362] (on-demand hierarchical), Bond [363] (on-demand flat), Condor [122] (computational flat), Darwin [364] (multimedia hierarchical), Globus [365] (various hierarchical cell), Legion [366] and Ninf [367] (computational hierarchical).

Similar to HPC clusters, it is also common to use centralized resource management in Grid computing where users could submit their jobs to a single resource management or scheduling entity that makes decision on which cluster and nodes run the job. Grids also deploy decentralized approaches for managing resources in the system.

However, there are some major drawbacks with the current decentralized resource management schemes in HPC clusters and Grids. Most of these approaches do not consider the dynamicity of the applications (e.g., the application load might change during the run-time) and the resources (e.g., resources might join, leave or fail) in computing environments and they statically pre-assign a group of resources to each resource manager (RM) entity in order to provide a load balanced solution to distribute jobs among different queues or RM entities in the system. This limits the dynamicity of resource management, since the resources are not allowed to be traded among different RM entities in the system (i.e., each resource is managed by a fixed RM entity). In other words, when a job is submitted to a RM, if the resources in RM resource-pool do not satisfy the query conditions the RM is not able to dynamically borrow resources from other RMs in the system and this results to either failing the query or forwarding the query to other resource manager entities in the system. Efficient decision making to forward queries among the distributed RMs leads to a resource discovery problem. Some of the decentralized RM approaches employ resource discovery techniques to manage query-forwarding between RMs, but these discovery mechanisms generally are not really dynamic and decentralized. For example Nimrod/G [39, 361] employs MDS resource discovery [368] which is based on publish/subscribe method to maintain resources in a set of hierarchical LDAP directories. MDS hierarchical discovery approach still suffers from single point failure mentioned for centralized methods because at each level there is a central data base server responsible for resource update requests. Moreover, MDS does not scale well in computing environments with frequent updates and large numbers of application requests.

2.7.3 Cloud Computing Systems

In Cloud computing systems, resource provisioning using on-demand virtualization and VM migration technologies enable the data centers to consolidate their computing services and use minimal number of physical servers. In fact, the virtualization allows workloads to be deployed and scaled-out quickly through the rapid provisioning of Virtual Machines (VMs) on physical machines. But similar to HPC cluster and Grid, Cloud computing systems still need to deal with resource allocation and resource discovery problem to efficiently map the virtual machines to the real physical resources.

An efficient resource management (in IaaS Cloud service model) can potentially provide significant number of benefits specifically in terms of quality of service, scalability, throughput and utility optimization, cost efficiency and overhead/latency reduction. However, for future diverse, large-scale systems, there are issues such as resource modeling (description), estimation

(identification), provisioning, discovery, brokering, mapping, allocation and adaptation that it should yet deal with.

Implicitly, the scale in Cloud systems must be regarded as horizontal where instances of data and code are replicated to increase availability. Whilst this is particularly useful for data and service hosts (eBay, Amazon, etc.), it does have little or no impact on scalable systems acting on vertical scale (i.e., where the actual application is split into multiple processes or threads that jointly contribute to the application's functionality). Recently there have been more and more references to the so-called HPC Cloud [369]. However these either focus on making small parallel machines (for instance, in the order of 8 cores in Amazon EC), or provide access to thousands of cores in an almost unconnected fashion, similar to Compute Grids, (like Plura Processing). In the current literature there are several types of resource management systems for Cloud computing. For instance we can mention OpenNebula [370], Eucalyptus [371], In-VIGO [372] and Cluster-on-Demand [373].

2.7.4 Manycore Systems

Current approaches for resource management in manycore systems can be categorized to off-line, mixed and on-line approaches. The off-line approaches such as [374–376] and [377] are based on priori (static) knowledge about the whole system states and resources, thus they can not resolve the unpredicted dynamic situations particularly when the application workload dynamically varies at run-time. The mixed approaches such as [378, 379] and [380] are based on the pre-calculation of all potential application mapping and selection of the best mapping at run-time. This also requires that all input applications and all possible combinations of mapping (for those applications) must be known at design-time. These approaches generally employ a central entity to select the best mapping amongst the pre-calculated solutions at run-time. But the application behavior and sequence of execution might be changed at run-time. Dynamic workload is not supported by mixed approaches. In online mapping (e.g., [381–383] and [384]), instead of pre-determination, the application mapping is dynamically decided at run-time. However we must note that online mapping trades off the resource allocation accuracy (i.e., the quality of resource mapping) versus computational complexity and scalability.

2.8 Conclusion

Resource discovery is one of the most significant challenging issues, particularly in large-scale distributed environments, which have already become mainstream platforms in academia and industry. It has a large variety of applications ranging from web-based resource discovery to job/task execution in high performance computing clusters, while it can be implemented on top of different computing environments. Thus, due to the objectives and the purposes of designing and implementing a resource discovery approach, it can be seen through a wide range of concepts, methodologies, design and evaluation aspects. In this chapter, we have discussed resource discovery solutions for large scale distributed environments, with special focus on the design aspects. Accordingly, we have categorized all of these aspects in three classes: underlying aspects, design aspects and evaluation aspects.

In underlying aspects, we have investigated and reviewed the current state of resource discovery protocols from the perspective of structure. We highlighted that the overall characteristics of computing environments (e.g., Grid, Cloud, HPC, HTC, Cluster, etc.) have a large

impact on most of the design and quality aspects (e.g., in terms of, objectives, methodology and even performance) of any potential proposal for resource discovery for those environments. Moreover, we presented the advantages and disadvantages of using different distributed architectures, such as centralized, hierarchical, decentralized and decentralized-tree, as underlying information structures. Additionally, we have reviewed the literature from the aspects of ontology and resource description models and different clustering approaches.

In design aspects, we have covered most methods and techniques for resource discovery in several aspects, including discovery strategies, search methods, packet propagation techniques, information delivery, synchronization and query termination. However, we did not cover security and summarization aspects. This provides a clear understanding about the potential methods, their weaknesses and strengths. Furthermore, we have reviewed the current works in the state of the art considering those methods.

In evaluation aspects, we have concluded that the most important evaluation and performance factors for resource discovery assessment can be classified in five different aspects including scalability, efficiency, reliability, flexibility and heterogeneity.

Furthermore, we can conclude that, referring to the current state of the art, to the best of authors knowledge, there is no extensive work in the literature, addressing specifically the problem of resource discovery (in thread-level) with respect to the various and complex requirements of future large dimension many-core enabled computing systems (such as high heterogeneity, scalability, dynamicity, efficiency, adaptability, querying flexibility in terms of complex querying, query expressiveness, resource graph discovery, etc).

We have also reviewed the literature on resource management approaches for different distributed systems at the end of this chapter. We can summarize that most of the existing resource management systems do not support fully decentralized, dynamical, scaleable, elastic and high quality resource mapping and allocation particularly for future large-scale manycore enabled computing systems with respect to multiple aspects such as changes in resources and the application behaviors. Workloads (i.e., the loads for applications) might be changed over time, thus the resource allocation for a process/application might need to grow or shrink in size. And also a process migration mechanism may required to tolerate failing/overloading resources. These are paradigms that are beyond most of the current resource management approaches in large-scale distributed computing.

Chapter 3

Hierarchical Resource Description

Resource discovery and resource mapping in large-scale many-core-enabled highly-heterogeneous computing environments require a scalable and expressive resource description model. In this chapter, we propose a hierarchical resource description and abstraction model to express the functionality of hardware (and the system architectures) in a scalable and semantic way.

3.1 Introduction

Hardware Description Languages (HDLs) [385, 386] do not allow for the productivity of hardware engineers to keep pace with the development of chip technology. While traditional HDLs, like VHDL [387] and Verilog [385], are very good at describing detailed hardware properties such as timing behavior, they are generally cumbersome in expressing the higher-level abstractions needed for today's large and complex circuit designs. With the increasing complexity in architecture design, it is preferable to lift the design process to a higher, more abstract level.

In a distributed computing environment, in addition to the structural and architectural information of the hardware resources and their relationships to each other (topology), the actual capabilities of the hardware are needed for resource discovery and mapping of code properties (i.e., to exploit the specific capabilities of the hardware through a scalable and expressive resource description model). In fact, we need to describe the hardware resource capabilities in a way that allows us to:

- Identify the requirements of the codes
- Do resource discovery and mapping
- Adapt the code to make the best use of the hardware capabilities

For this purpose, we must note that, describing all the computing resources in a large scale computing environment (e.g., HPC many-core system) with respect to efficiency and scalability, by employing a centralized design, cannot be applicable due to the potential issues of single point of failure and bottleneck. On the other hand, using distributed solutions such as Grid and P2P make us able to create hybrid systems, which have the advantages of both Grid and P2P. For example, in a common P2P system, generally, each one of the peers has to get information from other peers and disseminate the information to others through neighbor

peers. In this case, it is significant to consider the point where the resource information has to be distributed and get balanced between peers. It helps improving the scalability of the system when the number of peers increasingly grows. In this chapter we propose a new hardware description model that considers how the hardware resource information is distributed between computing-enabled peers (resources), and how the hardware resource capabilities can be described and derived from hardware descriptions.

The proposed hierarchical hardware modeling and resource description mechanism is able to abstract the characteristics and behavior of the large scale many-core-enabled computing systems in a way that both computational and communicational system properties are well represented to provide a close estimation of the real system, while avoiding to describe the hardware at a cycle-accurate level.

This model is both detailed and generic for resource discovery and mapping, while dealing with aspects such as capturing static and dynamic capabilities of hardware resources, and making distribution of hardware information scalable. We only focus on capturing the necessary information that will aid different algorithms along the line of resource discovery and mapping. However, we note that such a hardware description might not be detailed enough to actually derive a compiler, but it will be enough for our purpose.

3.2 Hierarchical Layers

The depth of hierarchy (i.e. number of layers in resource description model) and the definition of each layer might range from very high level (e.g. super clusters, clusters) to very low level (e.g. processing core, Arithmetic Logic Units (ALUs)) depending on architecture designing aspects. The layering is performed by partitioning the total number of n peers (i.e. resources) in the system into c disjoint cells of peers (i.e. c is the number of hierarchies in the distributed environment), in a way to ensure that peers in the same layer of each cell are similar to one another according to some predefined and inherited attributes. Each layer has two unique related layers: parent layer and child layer. A layer can be either a blank (null) layer with completely no information (definition) or it can be an identifiable layer with a deterministic definition. Furthermore, the properties of the peers in the upper layers are also inherited by the peers in the lower layers (see Figure 3.1).

We must note that the definition of a “resource” (i.e. computing resource) for our resource discovery approach might be highly variable depending on the levels of the described hierarchy. In fact, in our resource discovery system, we refer to a single resource as a representation of any individual peers, belonging to the lowest level of the resource description hierarchy. For example, we can define the ALU layer (i.e. the processing unit or micro-architecture layer) to describe the properties of computing resources in a low level layer with very detailed information. The ALU layer can also describe how the ALUs of a processor (CPU or core) gains access to data. In other words it has to decide on the number of the words to be used per cycle, how wide the words should be and whether the word sizes are configurable or not, etc. A resource can represent either a single ALU or a single core if the description model describes either the ALU layer or the Core layer accordingly as the lowest level of the hierarchy. Note that the granularity of the hierarchy will depend on the intended usage for the computing system.

We must consider that, in a distributed system it is required to employ an efficient data comparison and serialization method, which has a significant impact on the system data

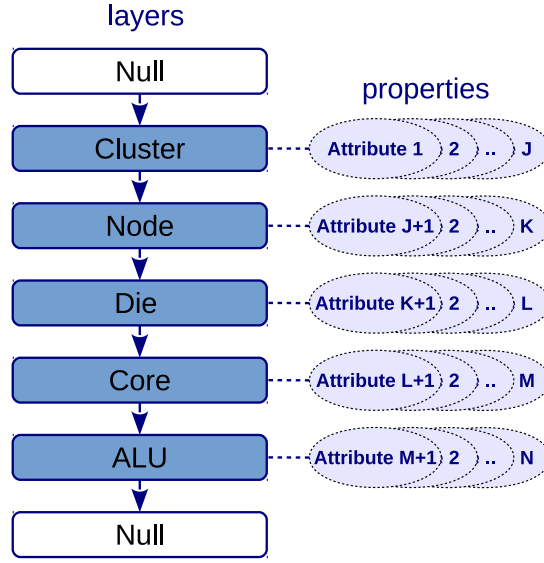


Figure 3.1: An example of description of resources in hierarchical layers.

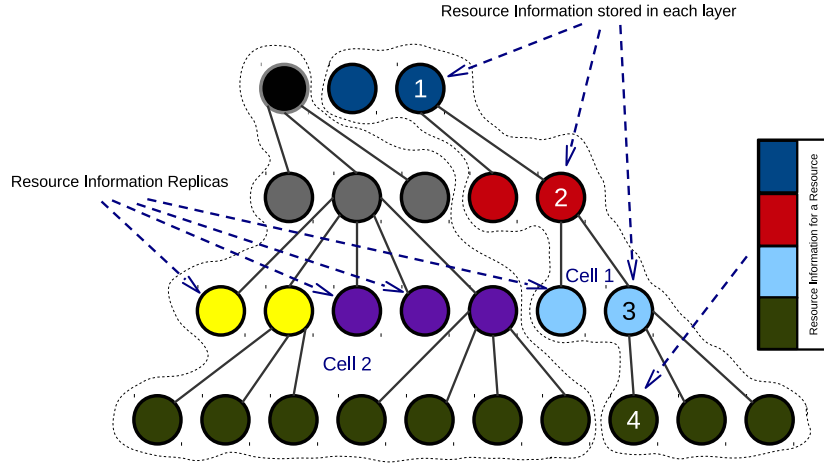


Figure 3.2: Distribution of resource information in a four-layers-hierarchy with 2 cells.

transmission, storage, and retrieval. Therefore the description should be as short as possible; however it may trade accuracy for shorter length.

According to the proposed resource description, we don't need to store locally all the information of a specific resource. The common resource information in each layer, instead of being repeated in all the peers in a local cell, is just maintained in a certain number of peers, which are offering resource information services to other nodes in the group or network. Figure 3.2 depicts the distribution of resource information in two individual cells (group of resources) where due to hierarchy, each cell in each layer stores common information of the members in the lower layers in certain peers, which are providing the information services to the others.

Table 3.1 demonstrates an example of attribute definition for a hierarchical resource description model, which consists of three layers: node (Inter-Node), processor (Inter-Dye) and

core (Inter-Core). For each layer, multiple attributes are defined by specifying mandatory fields such as attribute-ID, attribute-Name, attribute-Type and attribute-Value. The attribute-ID specifies a unique identifier for an attribute which can be globally used in the entire description system to refer an specific attribute. The attribute-Name assigns an acronym (short name) for each given attribute (an optional filed, like attribute-Description, can include longer field description). The attribute-Value contains the value for each attribute and the attribute-Type declares the type (Numerical, String or Bit String) of the value assigned.

Table 3.1: An example of layer definition with multi-dimensional attributes.

ID	Layer	Name	Value Type	Sample Value	Description
1	Inter-Core=1	AF	Bit String	And=bit0, OR=bit1, Not=bit2, XOR=bit3, Add=bit4, Sub=bit5, Mul=bit6, Div=bit7, ShiftL=bit8, ShiftR=bit9, Rotate=bit10, etc.	ALU Functionalities
2	Inter-Core=1	CL1	Numerical	128 Bytes	L1 Cache Line
3	Inter-Core=1	CL2	Numerical	128 Bytes	L2 Cache Line
4	Inter-Core=1	CA1	Numerical	2 Ways, Fully Associativity=0, Direct Mapped=1	L1 Cache Associativity
5	Inter-Core=1	CA2	Numerical	8 Ways, Fully Associativity=0, Direct Mapped=1	L2 Cache Associativity
6	Inter-Core=1	CCR	Numerical	1500 MHz	Core Clock Rate
7	Inter-Core=1	L1S	Numerical	128 KB	L1 Cache Size
8	Inter-Core=1	L1L	Numerical	4 ns	L1 Latency
9	Inter-Core=1	L2S	Numerical	524 KB	L2 Cache Size
10	Inter-Core=1	L2L	Numerical	13 ns	L2 Latency
11	Inter-Core=1	L3L	Numerical	30 ns	L3 Latency
12	Inter-Core=1	ML	Numerical	52 ns	Memory Latency
13	Inter-Core=1	MIP	Numerical	4500	MIPS
14	Inter-Core=1	NA	Numerical	4	Number of ALUs
15	Inter-Core=1	DPC	Numerical	6	Number of Data-PU Channels
16	Inter-Core=1	IPC	Numerical	4	Number of Instruction-PU Channels
17	Inter-Core=1	TA	Numerical	Integer=0, Float=1 (FPU), Both=2	Type of ALUs
18	Inter-Core=1	VL	Numerical	64 Bits	Vector Length
19	Inter-Dye =2	BF	Numerical	1333 MHz	BUS Frequency: Speed between CPU and RAM
20	Inter-Dye =2	MB	Numerical	1600 MB/S	Memory Bandwidth
21	Inter-Dye =2	MF	Numerical	166 MHz	Memory Frequency
22	Inter-Dye =2	CC	Numerical	None=0, Directory=1, Broadcast=2	Cache Coherence
23	Inter-Dye =2	ISA	Numerical	X86=0, SPARC=1, ARM=2, XCORE=3, RISC=4, CISC=5, Legacy=6, N/A=7	ISA
24	Inter-Dye =2	MA	Bit String	SMT=bit0, SuperScalar=bit1, VLIW=bit2, SIMD=bit3, In-Order=bit4, OutOfOrder=bit5	Micro Architecture
25	Inter-Dye =2	INT	Bit String	Bus=0, Ring=1, NOC=2, Crossbar=3, PointToPoint=4, HierarchicalNOC=5	Interconnection Network
26	Inter-Dye =2	ABS	Numerical	32 Bits	Size of Processor Address Bus
27	Inter-Dye =2	DBS	Numerical	64 Bits	Size of Processor Data Bus
28	Inter-Dye =2	PN	String	Intel core I7, AMD Phenom, ARM Cotext-A9, Intel Atom, Core 2 Duo, etc.	Processor Name
29	Inter-Dye =2	PC	Numerical	SISD=0, SIMD=1, MISD=2, MIMD=2	Processor Class
30	Inter-Dye =2	PT	Numerical	CPU=0, GPU=1, FPGA=2, ext.	Processor Type
31	Inter-Dye =2	L3S	Numerical	1024 KB	L3 Cache Size
32	Inter-Dye=2	CA3	Numerical	4 Ways, Fully Associativity=0, Direct Mapped=1	L3 Cache Associativity
33	Inter-Dye=2	CL3	Numerical	256 Bytes	L3 Cache Line
34	Inter-Dye =2	NC	Numerical	8	Number of Cores
35	Inter-Node =3	WS	Numerical	85 KB	Window Size
36	Inter-Node =3	MS	Numerical	4096 MB	Memory Size
37	Inter-Node =3	TNC	Numerical	56	Total Number of Cores or Nodes
38	Inter-Node =3	DC	Numerical	4	Die Count
39	Inter-Node =3	NB	Numerical	95 MB/S	Network Bandwidth
40	Inter-Node =3	NL	Numerical	273 ns	Network Latency

3.3 Information Transmission

As we need to make the information of the hardware architecture and capabilities (i.e., hardware description) available across computing nodes (for efficient code/application distribution) in heterogeneous large-scale dynamic environments, we need to categorize information relevant at the different hierarchy levels. As the systems are potentially highly dynamic, capability information might have to be transmitted frequently, imposing the need to minimize the overhead of the hardware resource description (or capability information) exchange. For this purpose, hierarchical information encoding makes sense, as we can reduce capability information being transmitted to that which is relevant for each specific level in the hierarchy.

We need to have a more abstract and high-level model of hardware description which includes useful, effective and short information that can be appropriate for resource discovery and resource mapping.

We must also note that, one of the primary performance aspect for any hardware description model is related to all forms of communication and data exchange between any two endpoints. It is thereby irrelevant whether the communication takes place between two code instances (threads), residing on two individual single cores, or between a code instance and a storage unit (i.e., between a single core processing unit and a memory unit such as cache or main memory). In all cases, the limiting performance factor is given by the hardware characteristics of the interconnects affected. This is closely related to the memory wall problem [388] that, in particular, multi-core systems are facing as multiple processing units will access the same physical memory and thus compete for cycle time and interconnect routes [389, 390].

Furthermore, the degree and types of communication potentially taking place in a system and for an application execution are manifold, ranging from register access over shared data to explicit communication. In all these operations, it is possible that the nodes (individual computing hardware) in the system, require to exchange their capability information (i.e., resource description). To do this, due to the potential network overhead and traffic, the data size (description size) and the transaction frequency of the description must be reduced as much as possible.

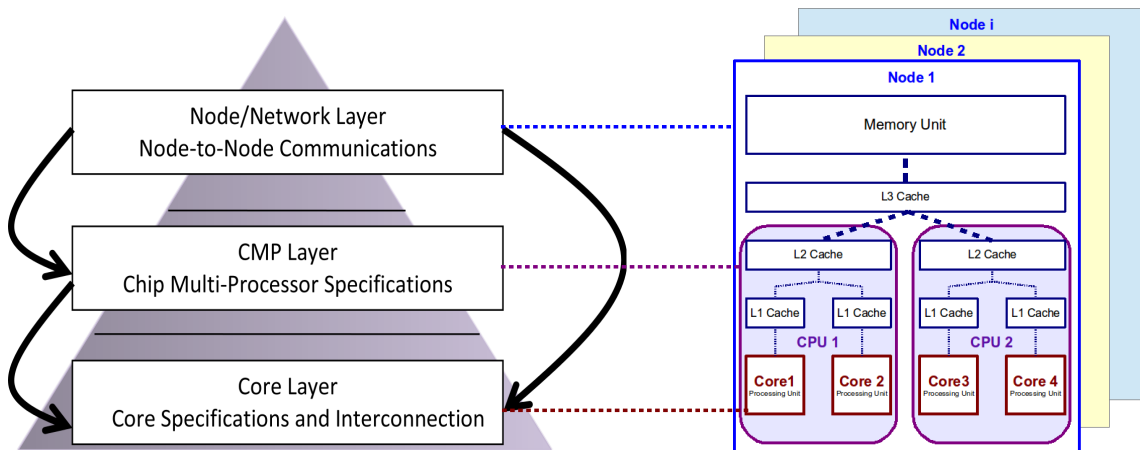


Figure 3.3: An example architecture of the hierarchical hardware description.

Figure 3.3 illustrates an example of the hierarchy model of our resource description system.

We categorize all relevant resource information, including computational and communication properties in different layers, going from more abstract information to more detailed characteristics. By using a hierarchical resource description model we can, at each level, just transmit the information required for the specific needs of a given discovery procedure. The model needs to rely on techniques for information aggregation and summarization. Information available at different domains will contain different levels of detail. Aggregation and summarization allow nodes to have coarse views about their neighbors, and then, after deciding that a neighbor is a potential candidate, get detailed information about its resources. For our model we consider the existence of the following layers:

- Layer1 (Inter-Core Level or NoC Level): Resource description on this level is characterized by information directly related to core internal operation. The most typical information in this level is related to the description of the cores, performance counters, private cache size or routing paths established inside the NoC. This information is of interest for optimizing the operation inside the chip and managing cores locally.
- Layer2 (Inter-Chip Level): This describes the specifications of the edges for the core communications, interconnection networks and memory hierarchy which, includes information such as bandwidth, latency, shared and/or global cache size, as well as the number of cores available and their characteristics. This information is relevant to the management of several boards in the same host. Traditional operating systems take into account this information when scheduling tasks in Symmetric Multi Processing (SMP) [391] systems.
- Layer3 (Inter-Board Level): Resource description here covers a total overview of all the communications, memory and processing features and aspects in the nodes, such as memory size and number of processors, and coarse performance counters, as well as inter-node communication information and its network structure. This aggregate information is available to other nodes to optimize task placement across hosts in the same operational domain (see Table 3.2).

Table 3.2: Examples of resource attributes in each layers.

Layers	Communication	Attributes
Layer3	Node2Node	Core Clock Rate, L1 Cache Size, L1 Latency, L2 Cache Size, L2 Latency, L1 Cache Line, L2 Cache Line, L1 Cache Associativity, L2 Cache Associativity, MIPS, Number of ALUs, Type of ALUs, ALU Functionalities, Number of Data-PU Channels, Number of Instruction-PU Channels, Vector Length, L3 Latency, Memory Latency, etc.
Layer2	Die2Die	BUS Frequency, Memory Bandwidth, Memory Frequency, Cache Coherence, ISA, Micro Architecture, Interconnection Network, Size of Processor Address Bus, Size of Processor Data Bus, Processor Name, Processor Class, Processor Type, Number of Cores, etc.
Layer1	Core2Core	Memory Size, Number of Dies, Network Bandwidth, Network Latency, Total Number of Cores, etc.

The distribution of resource information in the system is schematically depicted in Figure 3.4. According to this scheme, each node has the possibility of directly getting the highest level of information about itself and its close neighbors in different circles of vicinity. Furthermore,

each node has some summarized information about remote nodes, which is enough to provide an overall perspective of the whole system from its point of view. It is obvious that nearby resources with lower latency will be preferred in the discovery procedure. This is particularly relevant if the tasks which are to be distributed have input and/or output dependencies associated with the local devices. Task placement must take in consideration both processing capabilities and communication (I/O, network) latency.

We describe resources by using a set of attributes which are described in two different classes of individual and collective attributes. Individual attributes are the particular properties and characteristics for each resource. Collective attributes describe the communication properties between individual members inside a group of resources or group of resource groups. Link properties are necessary to create a descriptive graph of the interconnected resources, which can be employed for mapping between application segments and required set of resources. Attributes are static (such as Cache Size, Frequency, Processor Type, number of cores, etc.) or dynamic (such as CPU load, available memory, available bandwidth, etc.). The static attributes are not required to be updated in short intervals, however dynamic attributes are very sensitive and they might change very frequently.

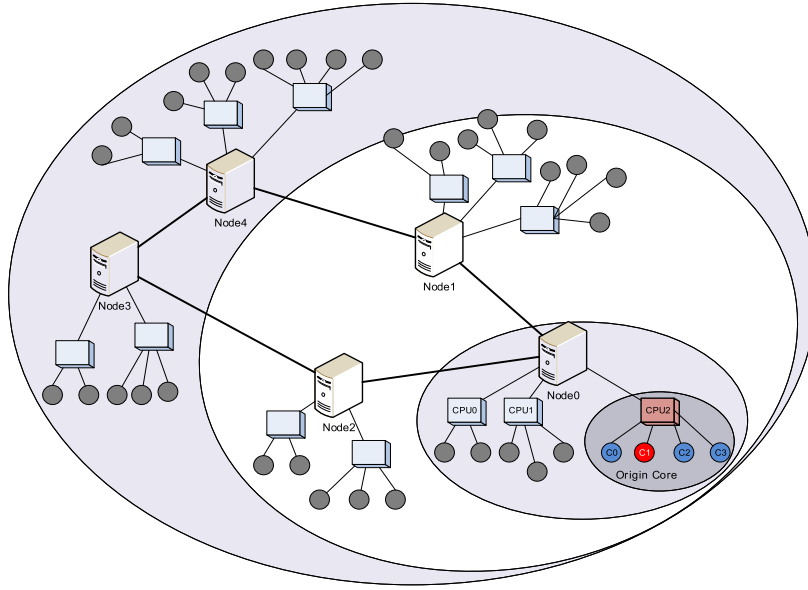


Figure 3.4: Fish eyes's model of distribution of the hardware capability information within the system.

We store the resource information discussed above in different layers, and in a ring-based distributed hash table where resources are placed in a ring according to their hashed keys. With these hashing functions we can map all attribute values in a specific layer to the same m -bit space in a DHT ring. We must consider that depending on the resource discovery approaches, it is not necessary to use DHT in all layers. Specifically for the upper layers we can store resource information in the format of the layer stamps, which can be validated according to a set of conditions in the incoming query templates. In our system we use different procedures at different layers by efficiency/complicity trade-offs.

3.4 Syntaxes and Description Examples

The proposed description model can flexibly describe various "systems" and "queries", ranging from very simple to very complex, while enabling scalable distribution of the resource information across the system. Here, we discuss the features and the syntax of our description language for describing attributes, layers, queries and systems. We store the definitions of layers and attributes in files with extension ".def". We also store the descriptions of queries and systems into files with extension ".des".

3.4.1 Defining Layers and Attributes

Grammar 3.1 presents the syntax, by Backus-Naur Form (BNF) grammar, for defining layers and attributes using our resource description language. All definitions required for a system design, including layers and attributes, must priorly be declared using the *definition* rule in a .def file. This can be done by using keyword *def*, followed by an *identifier* and sequences of *layer-definitions*. The syntax also supports two other alternative expressions in order to provide capability to define attributes independent of *layer-definitions*. These expressions include either *layer-definitions* as well as definition of other-attributes (i.e., independent attributes) or only definition of independent attributes. In fact, the definition of attributes can be either built-in (i.e., embedded in the layer-definition) or independent. The independent attributes are particularly used to describe queries where the inter-resource and inter-group communications required for a single group or multiple groups of resources need to be clearly specified in the query. They are specified by using keyword *attributes* at the beginning of a block.

The keyword *layer* is used to define a layer. A *layer-definition* consists of a list of *attribute-definitions* where each *attribute-definition* individually defines a built-in attribute for its corresponding layer. An *attribute-definition* can be specified by an unique *attribute-id* followed by an identifier, a short description of the attribute and the type of attribute. In addition, we can specify whether the defined attributes are static or dynamic, by using keywords *static* and *dynamic* in the body of the *layer-definition* and before defining each set of attributes (attributes are specified as static by default). Static attributes (e.g., CPU type) represent attributes that do not change at runtime. In contrary, dynamic attributes (e.g., CPU load) will potentially change at runtime. The type of each attribute can be *number* (positive integer), *byte* or *bitstring*. A *bitstring* is defined as a sequence of zero or more bits, by using keyword *bitstring* followed by a number in parentheses, where the number indicates the number of bits required. The type *bitstring* is useful to efficiently represents attributes which their values can be either one of multiple fixed-choices (i.e., *bitstring(number,value)*) or a combination of multiple fixed-choices (i.e., *bitstring(number,bit)*). For example we can specify the type of *AF* attribute (ALU Functionalities) for each core as *bitstring(11,bit)*, where each bit in the binary string (with length=11 bits) indicates whether a specific functionality is supported or not (e.g., "And"=bit0, "OR"=bit1, "Not"=bit2, "XOR"=bit3, "Add"=bit4, "Sub"=bit5, "Mul"=bit6, "Div"=bit7, "ShiftL"=bit8, "ShiftR"=bit9, "Rotate"=bit10). Another example would be the attribute INT which describes the topology of interconnection network for a processor. We can define the type of this attribute as *bitstring(3,value)* where each possible value of the attribute can represents a specific topology as listed as following: "Bus"=0, "Ring"=1, "NOC"=2, "Crossbar"=3, "PointToPoint"=4, "HierarchicalNOC"=5, "Others"=6.

Listing 3.2 presents a description example, using Grammar 3.1, which includes definitions of 3 layers: Layer1, Layer2 and Layer3. Each layer consists of definition of 2 to 4 individual

```

1 <Definition> ::= def <Identifier> <LayerDefinitionList> end
2       | def <Identifier> <LayerDefinitionList> <OtherAttributes> end
3       | def <Identifier> <OtherAttributes> end
4 <OtherAttributes> ::= attributes <AttributeDefinitionList> end
5 <LayerDefinitionList> := <LayerDefinition>
6       | <LayerDefinition> <LayerDefinitionList>
7 <LayerDefinition> ::= layer <Identifier> static : <AttributeDefinitionList> dynamic : <
   AttributeDefinitionList> end
8       | layer <Identifier> <Options> <AttributeDefinitionList> end
9       | layer <Identifier> <AttributeDefinitionList> end
10 <AttributeDefinitionList> ::= <AttributeDefinition>
11       | <AttributeDefinition> , <AttributeDefinitionList>
12 <AttributeDefinition> ::= [<AttributeID>] <Identifier> <AttributeDescription> : <
   AttributeType>
13 <Options> ::= dynamic | static
14 <AttributeType> ::= <Number> | byte | <BitString>
15 <BitString> ::= bitstring (<Number>, value) { <ValueDefinitionList> }
16       | bitstring (<Number>, bit) { <BitDefinitionList> }
17 <ValueDefinitionList> ::= <ValueDefinition>
18       | <ValueDefinition> , <ValueDefinitionList>
19 <BitDefinitionList> ::= <BitDefinition> | <BitDefinition> , <BitDefinitionList>
20 <ValueDefinition> ::= <Value> = <ValueDescription> | <ValueDescription> = <Value>
21 <BitDefinition> ::= bit<Number> = <BitDescription> | <BitDescription> = bit<Number>
22 <Number> ::= <Natural Number> ::= number
23 <AttributeDescription> ::= <String> ::= string
24 <Value> ::= <Number>
25 <ValueDescription> ::= <String>
26 <BitDescription> ::= <String>

```

Grammar 3.1: The grammar to define layers and attributes in .def files

built-in attributes. Each attribute definition includes attribute-id, attribute-name, attribute-description and attribute type. The description specifies a set of pre-defined potential values for the bitstring-type attributes (PT and ISA). The code also includes the definition of 4 independent attributes at the end of the *def-block* which define a set of inter-node and inter-group communication attributes to describe queries.

3.4.2 Defining Queries

Grammar 3.3 specifies the syntax for defining queries. A query is defined by using keyword *query*, followed by and *identifier* (the query name) and a *list-of-statements*. The syntax allows using 4 different *statements* including, *import*, *single-node-definition*, *homogeneous-group-definition* and *heterogeneous-group-definition* (statements are separated by using a ";").

The *import* statement is used to import all definitions of layers and attributes, presented in a ".def" file. These definitions (of layers and attributes) are used to specify values for the pre-defined attributes in the *query-block*.

A *single-node* specifies the lowest level of abstraction (for both query and system description) through describing an atomic entity, representing the smallest computing unit in the systems (i.e., we can use the notion of "resource" as an instance of a single-node where each resource can not be divisible into other sub-resources). In fact, we can define a *single-node* depending

```

1 def "in-a.def"
2   layer Layer1
3     [1] CCR "Core Clock Rate by MHz" : number,
4     [2] L1S "L1 Cache Size by KB" : number,
5     [3] L1L "L1 Latency by ns" : number,
6     [4] L2S "L2 Cache Size by KB" : number
7   end
8   layer Layer2
9     [5] PT "Processor Type" : bitstring(2,value) {"CPU"]=0, "GPU"]=1, "FPGA"]=2, "Others"]=3} ,
10    [6] ISA "Instruction Set Architecture" : bitstring(3,value) {"X86"]=0, "SPARC"]=1, "ARM"]=2,
    "XCORE"]=3, "RISC"]=4, "CISC"]=5, "Legacy"]=6, "Others"]=7}
11  end
12  layer Layer3
13    [7] MS "Memory Size by MB" : number,
14    [8] DC "Die Count" : number
15  end
16  attributes
17    [9] INB "Inter-node Bandwidth Mb/S" : number,
18    [10] INL "Inter-node Latency by ns" : number,
19    [11] IGB "Inter-group Bandwidth Mb/S" : number,
20    [12] IGL "Inter-group Latency by ns" : number
21  end
22 end

```

Listing 3.2: An example of definitions for layers and attributes in .def files

on the level of abstraction required for querying. For example, a *single-node* can be defined in ALU-level, core-level, CPU-level or node-level. A *single-node* definition contains characteristics required for a single-node by specifying the desired *attribute-value(s)* for a subset of attributes in each pre-defined layer. In other words, a *single-node* definition is a single specification of query requirements for the smallest computing entity in the system. Furthermore, multiple *single-node* definitions (by using different unique identifier for each *single-node*) are allowed in order to precisely describe all requirements of a query. The language also supports various expressions and operators (such as parentheses, and, or, <, >, <=, >= and =) to specify value(s) for each attribute. We must note that for query description it is not necessary to specify required values for all pre-defined attributes of each layer, rather, depending on the query requirements, conditions for a subset of attributes might need to be specified (non-specified attributes are ignored during query processing). The keyword *all* can be used for layers that do not include any specific attribute conditions.

A *homogeneous-group* is defined as a set of *single-node* instances (resources) of a same type (specified by the *single-node-identifier*) with an optional indication of query requirements for communication among single-node instances (resources or group members). In order to define a *homogeneous-group* we can use keyword *homogroup*, followed by an identifier and both *group-size* and *single-node-identifier* (separated by a "," and in parentheses). The *group-size* specifies the number of members (resources) in the group. The *single-node-identifier* specifies the type for all members. We can also use keyword *inconstraints* to specify *inter-resource-constraints* if it is required (for a query) to provide conditions for communication between group members. The definition of *inter-resource-constraints* includes the specification of conditions for independent attributes (as discussed in Section 3.4.1). Alternatively, the keyword *homogroups* can be used to define multiple *homogeneous-groups* in a single-block,


```

1 <QueryDefinition> ::= query <Identifier> <StatementList> end
2 <StatementList> ::= <Statement> ; <StatementList> | <Statement> ;
3 <Statement> ::= <Import>
4         | <SingleNodeDefinition>
5         | <HomoGroupDefinition>
6         | <HeteroGroupDefinition>
7 <Import> ::= definition <String>
8 <SingleNodeDefinition> ::= singlenode <SingleNodeIdentifier> <LayerList> end
9 <LayerList> ::= <LayerExpression> | <LayerExpression> ; <LayerList>
10 <LayerExpression> ::= <LayerName> : <AttributeList> | <LayerName> : all
11 <AttributeList> ::= <AttributeExpression> | <AttributeExpression> , <AttributeList>
12 <AttributeExpression> ::= ( <AttributeExpression> )
13         | <AttributeExpression> and <AttributeExpression>
14         | <AttributeExpression> or <AttributeExpression>
15         | <AttributeName> <Operator> <AttributeValue>
16 <Operator> ::= < | > | <= | >= | =
17 <AttributeName> ::= <Identifier>
18 <SingleNodeIdentifier> ::= <Identifier>
19 <AttributeValue> ::= <Integer> | <Byte> | <Const> | <BitString>
20 <HomoGroupDefinition> ::= homogroup <HomoGroupExpression> inconstraints : <AttributeList> end
21         | homogroup <HomoGroupExpression> end
22         | homogroups <HomogroupExpressionList> end
23 <HomoGroupExpressionList> ::= <HomoGroupExpression> | <HomogroupExpression> , <
        HomogroupExpressionList>
24 <HomoGroupExpression> ::= <HomoGroupIdentifier> (<GroupSize> , <SingleNodeIdentifier>)
25 <HeteroGroupDefinition> ::= heterogroup <Identifier> ( <HomoGroupIdentifierList> )
        igconstraints : <AttributeList> end
26         | heterogroup <Identifier> ( <HomoGroupIdentifierList> ) end
27 <HomoGroupIdentifierList> ::= <HomoGroupIdentifier> | <HomoGroupIdentifier> , <
        HomogroupIdentifierList>
28 <GroupSize> ::= <Integer>

```

Grammar 3.3: The grammar to define queries in .des files

whenever the specification of inter-resource conditions for those groups are not required by the given query.

Similarly, a *heterogeneous-group* can be defined as a set of *homogeneous-groups* with an optional specification of query conditions for communication among *homogeneous-groups* (inter-group constraints). A definition of a *heterogeneous-group* generally includes keyword *heterogroup*, followed by an identifier (i.e., the *heterogroup* name) and a list of *homogroup-identifiers* (names of *homogroup* members in parentheses and separated by a ","). Conditions for communication between *heterogroup* members might be included in the definition of a *heterogroup*. For these conditions, each *homogeneous-group* can be represented by a random member of the group. In other words, the *inter-group-constraints* specify the desired query requirements for communication between each pair of (homogeneous) group-representatives. This can be done by using keyword *igconstraints*, followed by specification of conditions for the required independent attributes.

Listing 3.4 demonstrates a simple query example that shows a query expression to search for 5 CPU cores with frequency rate of 2000 MHz, L1 cache size of 256 MB, L2 cache size of 512 MB and L1 cache latency of 8 ns. It also indicates that the range of communication latency between the requested resources must be between [20, 130] ns. This example describes a group of homogeneous resources (*HOGroup1*) with indication of inter-resource communication

requirements.

```
1 query Query1
2   definition "in-a.def"; // Importing the definition of attributes
3   singlenode SingleNode1 // Single resource
4     Layer1: CCR=2000, L1S=256, L1L=8, L2S=512;
5     Layer2: PT=0; // PT: Processor Type="CPU"
6     Layer3: all; // "*" No Query Constraints for Layer3
7   end;
8   homogroup HOGroup1(5,SingleNode1) // Homogeneous group of resources
9     inconstraints: (INL>=20) and (INL<=130); //Inter-node communication constraints
10  end;
11 end
```

Listing 3.4: A query expression example for a homogeneous set of resources in a 3 layer hierarchy

We can also define a heterogeneous group of resources by creating several homogeneous groups (each homogeneous group might have one or several members) and describing the query requirements for communication between those (homogeneous) groups (each homogeneous group can be represented by a random group member). In Listing 3.5, we have added another homogeneous group (*HOGroup2*) which contains 8 GPU cores with a set of specific attributes in each layer and then in the last part we have created a heterogeneous group by describing the query requirements for communication between groups (*HOGroup1* and *HOGroup2*).

```
1 query Query2
2   definition "in-b.def";
3   singlenode SingleNode1 // A single resource
4     Layer1: CCR=2000, L1S=256, L1L=8, L2S=512;
5     Layer2: PT=0; // PT: Processor Type="CPU"
6     Layer3: all; // "*" No Query Constraints for Layer3
7   end;
8   homogroup HOGroup1(5,SingleNode1) // A homogeneous group of resources
9     inconstraints: (INL>=20) and (INL<=130);
10  end;
11  singlenode SingleNode2 // A single resource
12    Layer1: CCR=1000, L1S=512, NA=4, TA=0, L2L=15;
13    Layer2: PT=1, PC=2, ISA=3; // PT: Processor Type="GPU", see Table 3.1 for more details of
        other attributes
14    Layer3: WS=90;
15  end;
16  homogroup HOGroup2(8,SingleNode2) // A homogeneous group of resources
17    inconstraints: (INL>=20) and (INL<=50);
18  end;
19  heterogroup HEGroup1(HOGroup1,HOGroup2) // A heterogeneous group of resources
20    igconstraints: (INL>=80) and (INL<=550); //Inter-group constraints
21  end;
22 end
```

Listing 3.5: A query expression example for a heterogeneous set of resources in a 3 layer hierarchy (see Table 3.1 for definition of attributes)

3.4.3 Defining Systems

In our approach, for a Distributed Operation System (DOS), the resource information for every single resources in the system can be extracted from the "system description" by using a component called "resource description provider". These information in turn are encapsulated into layer-stamps (see Section 3.5) and distributed among resources in different levels of hierarchy. Grammar 3.6 presents the syntax for defining systems. As we can see in the grammar, the syntax used here is identical to the one used for defining queries (see Grammar 3.3) with the following exceptions:

- The keyword *system* is used to define a system.
- The only supported operator within the *attribute-expression* is the equal sign ("="), as each attribute of a single-node must provide the exact value for the corresponding attribute.
- In order to precisely describe a system, it is desirable to specify *attribute-values* for most of attributes of each single node in the system. But if this is not feasible, the keyword *none* can be used within a *layer-expression* to highlight that none of attribute-values for the given layer are specified in the description.
- Hierarchies, required for describing different systems, can be built using multiple homogeneous and heterogeneous group definitions. But, unlike query description, the syntax for system description allows heterogeneous groups to be consisted of both homogeneous and heterogeneous groups.

In the remaining of this section we provide two description examples, expressing capabilities of different systems.

3.4.3.1 Micro-Architectures

Many contemporary architectures can be configured (through their instructions) to make use of the available resources (ALUs) in a very different way. As an example we can take the configurable SIMD Processing Unit (PU) shown in Figure 3.5a. When all ALUs are considered separately, the PU is a 4-wide SIMD machine where each ALU has 32 bit single-precision operands. However, when performing a double precision floating point operations, ALUs will operate together and will use 64-bit operands, resulting in a 2-wide SIMD machine. As we can see in the Figure 3.5a, it is also feasible that all ALUs are grouped together to perform an extended-precision operation of 128 bits, thus being a Single Instruction, Single Data (SISD) machine for such an operation.

We can expose the number of (virtual) ALUs depending on the instruction type. We can subsequently determine the Flynn category of a PU and the possible SIMD-width. Therefore, we can capture both the SIMD and SISD nature of this configurable PU by being able to specify the alternative hardware descriptions. Listing 3.7 provides description corresponding to the architecture demonstrated in Figure 3.5a.

The SIMD-PU has two arguments, which describe the inter-ALUs constraints. These constraints are the instruction-line (il0) and instruction-type (Single-Precision, Double-Precision or Extended-Precision), which lead to return a processing unit consisting out of one more

```

1 <SystemDefinition> ::= system <Identifier> <StatementList> end
2 <StatementList> ::= <Statement> ; <StatementList> | <Statement> ;
3 <Statement> ::= <Import>
4           | <SingleNodeDefinition>
5           | <HomoGroupDefinition>
6           | <HeteroGroupDefinition>
7 <Import> ::= definition <String>
8 <SingleNodeDefinition> ::= singlenode <SingleNodeIdentifier> <LayerList> end
9 <LayerList> ::= <LayerExpression> | <LayerExpression> ; <LayerList>
10 <LayerExpression> ::= <LayerName> : <AttributeList> | <LayerName> : none
11 <AttributeList> ::= <AttributeExpression> | <AttributeExpression> , <AttributeList>
12 <AttributeExpression> ::= <AttributeName> = <AttributeValue>
13 <AttributeName> ::= <Identifier>
14 <SingleNodeIdentifier> ::= <Identifier>
15 <AttributeValue> ::= <Integer> | <Byte> | <Const> | <BitString>
16 <HomoGroupDefinition> ::= homogroup <HomoGroupExpression> end
17           | homogroups <HomogroupExpressionList> end
18 <HomoGroupExpressionList> ::= <HomoGroupExpression> | <HomoGroupExpression> , <
           HomogroupExpressionList>
19 <HomoGroupExpression> ::= <HomoGroupIdentifier> (<GroupSize> , <SingleNodeIdentifier>)
20 <HeteroGroupDefinition> ::= heterogroup <HeteroGroupIdentifier> (<GroupIdentifierList>) end
21           | heterogroup <HeteroGroupIdentifier> (<GroupIdentifierList>)
22 <GroupIdentifierList> ::= <HomoGroupIdentifierList> | <HeteroGroupIdentifierList>
23 <HomoGroupIdentifierList> ::= <HomoGroupIdentifier> | <HomoGroupIdentifier> , <
           HomoGroupIdentifierList>
24 <HeteroGroupIdentifierList> ::= <HeteroGroupIdentifier>
25           | <HeteroGroupIdentifier> , <HeteroGroupIdentifierList>
26 <GroupSize> ::= <Integer>
27 <HomoGroupIdentifier> ::= <HeteroGroupIdentifier> ::= <String>

```

Grammar 3.6: The grammar to define systems in .des files

ALUs. The number of visible ALUs depends on the instruction type arriving on the instruction line, single-precision give four distinguishable ALUs, double-precision two ALUs, and extended-precision one ALU. This does not mean that the processing unit has 7 ALUs, but that the number of effective ALUs changes with the instruction type. We can determine the width of the operands by looking at the width of the data-lines (dl) connecting an ALU to the data pool.

3.4.3.2 Memory-Hierarchy

In Figure 3.5b, we depict a simple dual core setup where the cores share the main memory and L2 cache, but not L1 and the processors also do not support write / read through, so that any access to the main memory has to be routed via L1 and L2 (red arrow). The properties of the routed access can thereby be simply derived as follows:

$$bw \xrightarrow{from:src}{to:dest} = \min \left\{ bw \xrightarrow{from:src}{to:U_1}, bw \xrightarrow{from:U_1}{to:U_2}, \dots, bw \xrightarrow{from:U_i}{to:dest} \right\} \quad (3.1)$$

$$lat \xrightarrow{from:src}{to:dest} = lat \xrightarrow{from:src}{to:U_1} + lat \xrightarrow{from:U_1}{to:U_2} + \dots + lat \xrightarrow{from:U_i}{to:dest} \quad (3.2)$$

Our hardware description model is able to provide description for the memory hierarchy demonstrated in Figure 3.5b (see Listing 3.8). This model is however addresses the aspects

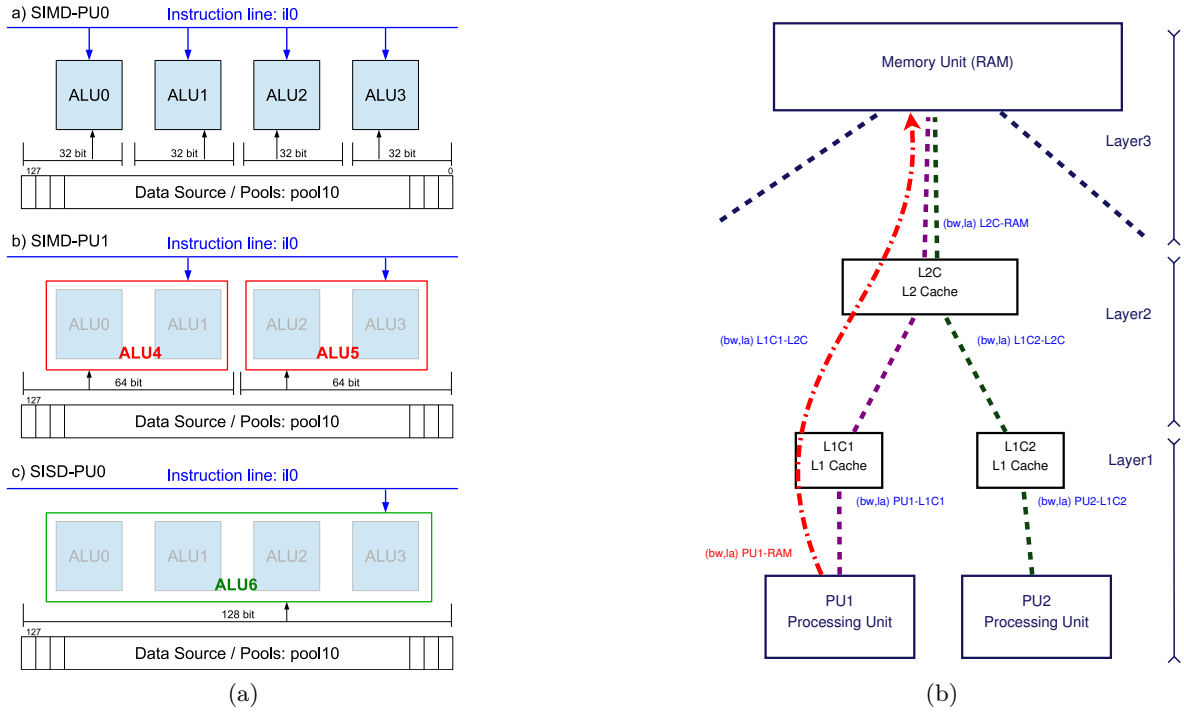


Figure 3.5: (a) Configurable SIMD processing unit. (b) A simple dual core setup (bw: bandwidth; la: latency).

arising from shared communication usage, such as when PU1 and PU2 access the RAM at the same time. In this case, bandwidth is reduced and latency may increase in Figure 3.5b. Notably, however, any such effect must be considered as a purely worst case scenario with increasing probability denoted by the density of communication and number of communicating instances.

```

1 system Micro-System1
2   definition "in-c.def";
3   singlenode ALU0
4     Layer0: aluType=FP, dataLine=dl0, dlPool=pool10,
5             dlAddress=0, dlWidth=32, dlpSize=128, dlpID=0;
6     Layer1: instrLine=il0, instrType=SP; //Single-Precision;
7   end;
8   singlenode ALU1
9     Layer0: aluType=FP, dataLine=dl1, dlPool=pool10,
10            dlAddress=32, dlWidth=32, dlpSize=128, dlpID=0;
11    Layer1: instrLine=il0, instrType=SP; //Single-Precision;
12  end;
13  singlenode ALU2
14    Layer0: aluType=FP, dataLine=dl2, dlPool=pool10,
15            dlAddress=64, dlWidth=32, dlpSize=128, dlpID=0;
16    Layer1: instrLine=il0, instrType=SP; //Single-Precision;
17  end;
18  singlenode ALU3
19    Layer0: aluType=FP, dataLine=dl3, dlPool=pool10,
20            dlAddress=96, dlWidth=32, dlpSize=128, dlpID=0;
21    Layer1: instrLine=il0, instrType=SP; //Single-Precision;
22  end;
23  singlenode ALU4
24    Layer0: aluType=FP, dataLine=dl4, dlPool=pool10,
25            dlAddress=0, dlWidth=64, dlpSize=128, dlpID=0;
26    Layer1: instrLine=il0, instrType=DP; //Double-Precision;
27  end;
28  singlenode ALU5
29    Layer0: aluType=FP, dataLine=dl5, dlPool=pool10,
30            dlAddress=64, dlWidth=64, dlpSize=128, dlpID=0;
31    Layer1: instrLine=il0, instrType=DP; //Double-Precision;
32  end;
33  singlenode ALU6
34    Layer0: aluType=FP, dataLine=dl6, dlPool=pool10,
35            dlAddress=0, dlWidth=128, dlpSize=128, dlpID=0;
36    Layer1: instrLine=il0, instrType=EP; //Extended-Precision;
37  end;
38  homogroups
39    HOGGroup0(1,ALU0), HOGGroup1(1,ALU1), HOGGroup2(1,ALU2), HOGGroup3(1,ALU3),
40    HOGGroup4(1,ALU4), HOGGroup5(1,ALU5), HOGGroup6(1,ALU6);
41  end;
42  heterogroup SIMD-PU0(HOGGroup0, HOGGroup1, HOGGroup2, HOGGroup3);
43  heterogroup SIMD-PU1(HOGGroup4, HOGGroup5);
44  heterogroup SISD-PU0(HOGGroup6);
45 end

```

Listing 3.7: A description example for configurable SIMD processing units

```

1 system System1
2   definition "in-d.def";
3   singlenode PU1
4     Layer1: CCR=val1, L1S=l1cv1, L1L=l1lat1, L2L=l2lat1, ML=meml1, MB=memb1;
5     Layer2: L2S=l2cv;
6     Layer3: MS=mems;
7   end;
8   singlenode PU2
9     Layer1: CCR=val2, L1S=l1cv2, L1L=l1lat2, L2L=l2lat2, ML=meml2, MB=memb2;
10    Layer2: L2S=l2cv;
11    Layer3: MS=mems;
12  end;
13  homogroups
14    HOGGroup1(1,PU1), HOGGroup2(1,PU2);
15  end;
16  heterogroup Config1(HOGGroup1,HOGGroup2);
17 end

```

Listing 3.8: An example of memory hierarchy description containing static and dynamic characteristics

3.5 Coding Efficiency

Referring to the proposed three layers (node-die-core) description model (in Section 3.3), we can describe the relevant hardware capabilities (i.e., the general system attributes) by defining arbitrary number of attributes per each layer. All attributes of each layer as well as their values must be represented by a single layer-stamp, which is the concatenation of the hex-decimal values of the attributes. These values are arranged and sorted within the layer-stamp based on the ordering of their attribute identifiers mentioned in the attribute-string (atr_{str}). The attribute-positioning (atr_{pos}) specifies the size of each attribute value correspondent to each attribute identifier in the attribute-string.

The layer-stamp for each layer can be constructed by concatenation of values (binary or hexadecimal values) of its pre-defined attributes with respect to the attribute ordering mentioned in the corresponding atr_{str} . The resulted layer-stamp can also be decoded by having the values for atr_{str} and atr_{pos} . For example, if we assume that a $layer_i$ has described by using 3 different attributes and their values (such as attribute8: id=8, length=16, value=8192_{dec}=2000_{hex}; attribute9: id=9, length=8, value=5_{dec}=05_{hex} and attributeA: id=A, length=16, value=82_{dec}=0052_{hex}), the layer-stamp of $layer_i$ for atr_{str} =89A is attribute8 + attribute9 + attributeA = 2000050052_{hex} with atr_{pos} =F7F_{hex}. Similarly, the layer-stamp of $layer_i$ for atr_{str} =A89 is attributeA + attribute8 + attribute9 = 0052200005_{hex} with atr_{pos} =FF7_{hex}.

Depending on the number of attributes in each layer, and the required space to store each attribute-value, the layer-stamp can be longer and consequently need more memory to be stored. Therefore, it is necessary to encode the layer-stamp using a low-cost, efficient encoding mechanism, which can reduce the length of the layer-stamp as much as possible. The expected encoding algorithm must be loss-less and low-cost, with high rate of reduction. Huffman coding [392] is one of the well-known classic methods to encapsulate data in a way that allows the original data to be perfectly reconstructed from the compressed data (i.e., loss-less data compression) [393–395]. However, the original algorithm might create long-length code-words, which decrease the rate of reduction, and also increase the cost of encoding by larger Huffman tables. We must note that in comparison to Huffman-encoding, other loss-less algorithms such as Run-Length encoding [396], Arithmetic coding [397], Context Tree Weighting [398] and Burrows Wheeler Transform [399] might provide better reduction rate. However, they consume more memory to maintain the required information for decoding, or they require a slower algorithm for data decoding.

In our work, in order to provide an efficient low-cost encoding algorithm, we employ a variation of Length-Limited Huffman (LLH) algorithm [400]. Figure 3.6 and Figure 3.7 demonstrate the procedures of encoding and decoding hardware resource information in different steps. For each layer, we first, construct the layer-stamps by hexadecimal encoding of the attribute-values, while considering the order of attributes in atr_{str} . In the second step, we encode the layer-stamps using LLH, where the maximum number of symbols is defined as 16 and also the maximum length of each code-word is 16 bits. Using this scheme, we would be able to encode unlimited number of attribute-values, specially for the systems that support large number of attributes per layer.

Figure 3.8 illustrates the bit-string template that we use to store the LLH coding information (coding-info-string) which consists of symbol, length of the code-word for the symbol and the code-word of the symbol, for the succession of different symbols in the given hexadecimal layer-stamp.

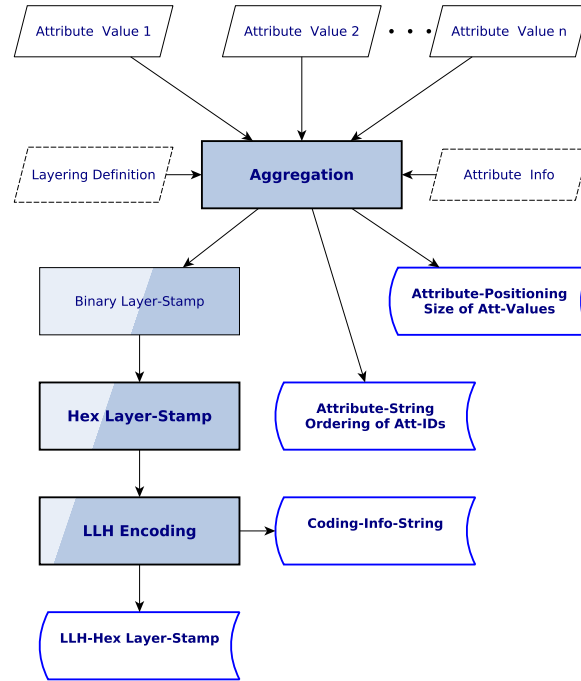


Figure 3.6: Aggregation procedure.

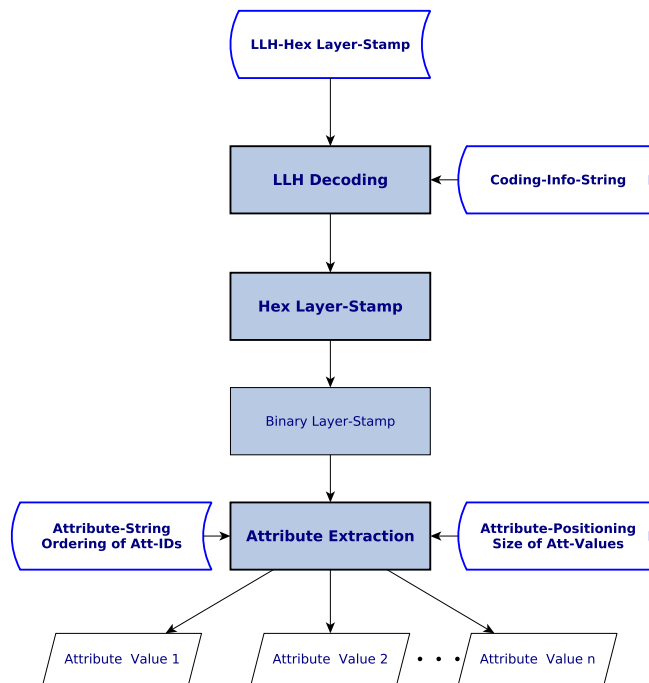


Figure 3.7: Attribute extraction procedure.

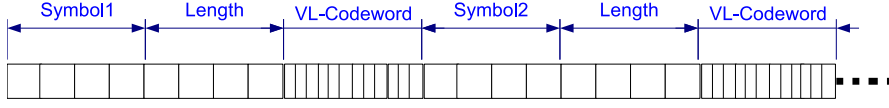


Figure 3.8: Binary string template for LLH-Hex encoding.

In fact, a LLH-Hex layer-stamp can be decoded using the aforementioned coding-info-string, bearing in mind that according to our defined coding-info-string, the constant length to store each different symbol is 4 bits, and the maximum length of each variable-length code-word is 16 bits. After decoding a LLH-Hex layer-stamp, the resulting hexadecimal layer-stamp can be translated to the attribute-values for each individual attribute in the layer by using the information encapsulated within atr_{str} and atr_{pos} .

In remaining of this section, we show how our resource description model is able to efficiently describe hardware capability information of the individual peers in a system, while it provides an efficient underlying scheme for information transmission and communication between peers during resource discovery. In our example, we assume that we can describe the relevant hardware capabilities, using 4 different attributes per layer. For this purpose, we select and define the general system attributes using Table 3.3. This table also provides the values of the described attributes for a sample single resource.

As it is shown in Table 3.3, atr_{str} for the $layer_1$ is 0123, which means that the layer-stamp can be constructed by concatenation of values (binary or hexadecimal values) of attribute 0, 1, 2 and 3. The resulting layer-stamp can be decoded by having the values of two parameters: attribute-positioning ($FFF7$) and attribute-string (0123). Thus, in order to extract the values of different attributes from the given layer-stamp, the binary stamp, according to atr_{pos} , must be divided to four individual parts with the size of 16, 16, 16 and 8 bits (according to the attribute definitions in Table 3.3) which are correspondent to the attribute 0, 1, 2 and 3, respectively.

Table 3.3: An example description of general system attributes as well as the capability information for a sample single resource.

ID	Att	Lay	Unit	Type	Len _{bits}	Value Range	Sample Value	Hex Value	atr_{str}	atr_{pos}
0	CCR	1	MHz	num	16	0-65,535	2000	07D0	0123	FFF7
1	L1S	1	KB	num	16	0-65,535	2048	0800	0123	FFF7
2	L1L	1	NS	num	16	0-65,535	150	0096	0123	FFF7
3	NA	1	—	num	8	0-255	4	04	0123	FFF7
4	NC	2	—	num	16	0-65,535	8	0008	4567	F333
5	PT	2	—	bit	4	0-15	CPU=0	0	4567	F333
6	INT	2	—	bit	4	0-15	Ring=1	1	4567	F333
7	ISA	2	—	bit	4	0-15	ARM=2	2	4567	F333
8	MS	3	MB	num	16	0-65,535	8192	2000	89AB	F7FF
9	DC	3	—	num	8	0-255	5	05	89AB	F7FF
10	TNC	3	—	num	16	0-65,535	82	0052	89AB	F7FF
11	NB	3	MB/s	num	16	0-65,535	100	0064	89AB	F7FF

Figure 3.9 illustrates the proposed LLH tree encoding layer1 information, which is represented by the layer-stamp 07D00800009604.

As we can see in Figure 3.9, the LLH encoding algorithm gets the hexadecimal string 07D00800009604 (the input string equivalent to a binary string with length 56 for the layer-stamp of Layer1) and generates an encoded binary string with length 30 as output. The

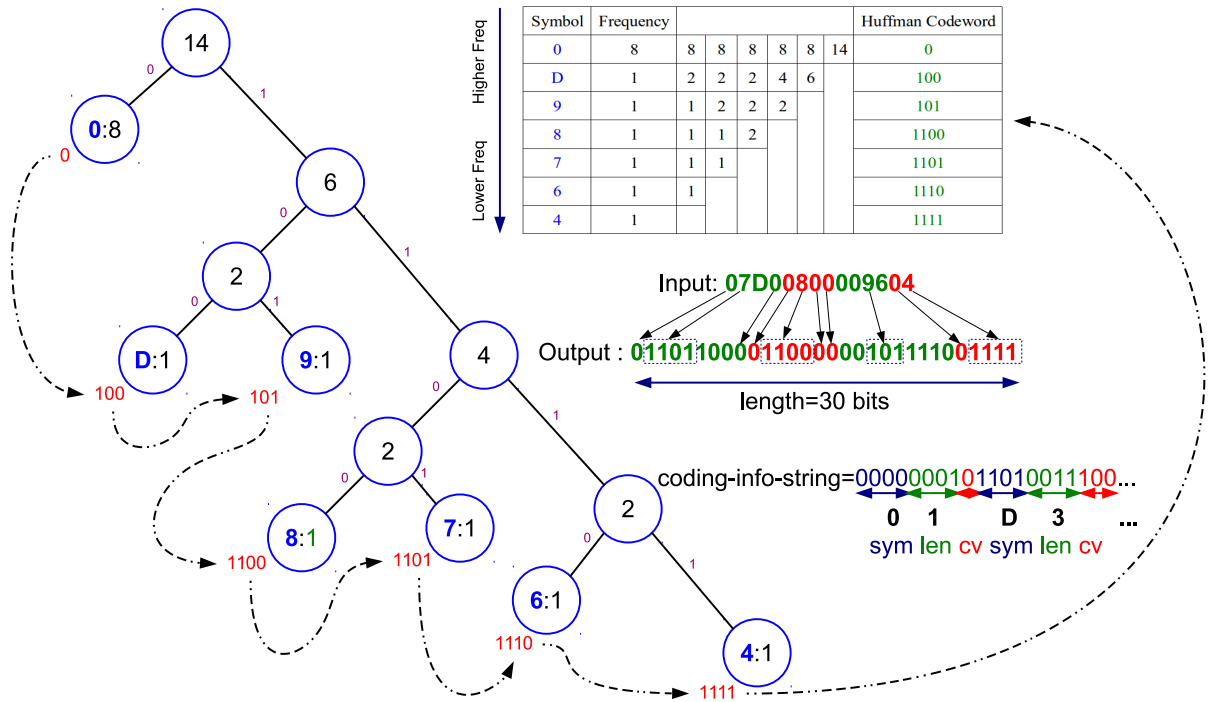


Figure 3.9: LLH encoding.

proposed LLH algorithm assumes that the input string is a hexadecimal string and each symbol is defined as a (fixed-length) single hexadecimal character presented in the input string. The algorithm, in the first step, sorts the list of symbols (for the input string) by frequency of each symbol in the input string (from higher frequencies to lower frequencies). In the next step, a tree is generated by making the two lowest elements (in the list) into leaves and creating a parent node with a frequency that is the sum of frequencies for those leaves. Afterwards, those two elements are removed from the list and the new parent node, with an accumulated frequency of f ($f = frequency(element1) + frequency(element2)$), is inserted into the list (sorted by frequency). This procedure is repeated by combining the two lowest elements in each iteration. The loop is repeated until there is only one element left in the list. The last element in the list becomes the root of the LLH tree. In the next step, the algorithm generates the LLH code for each symbol by traversing the tree from the root to the given symbol, outputting a 0 every time a left-hand branch is taken, and a 1 every time a right-hand branch is visited. The output string finally can be created by replacing each symbol in the input string with its corresponding LLH code, calculated by the algorithm. The other output of the algorithm is a coding-info-string which is used at the time of LLH decoding. This string records the coding tree, including list of symbols, LLH codes for symbols as well as the code-lengths. Decoding also can be done by reading bits from the input stream and traversing the code tree (starting from at the root for each code; taking the left-hand path if reading a 0 and the right-hand path if reading a 1). Once a leaf is hit, the symbol (for the traversed code) is found.

Figure 3.10 demonstrates a combination of Run-Length and Huffman (RLH) encoding algorithms. In Run-Length encoding, sequences in which the same value occurs in many

consecutive data elements are replaced with the run length and the data value. This approach can be combined with Huffman encoding by specifying each symbol as a variable-length sequence of repeated element. In the example shown in Figure 3.10, sequences such as 00, 07 and 04 are specified as single Huffman symbols. The rest of encoding process is similar to the example presented in Figure Figure 3.9.

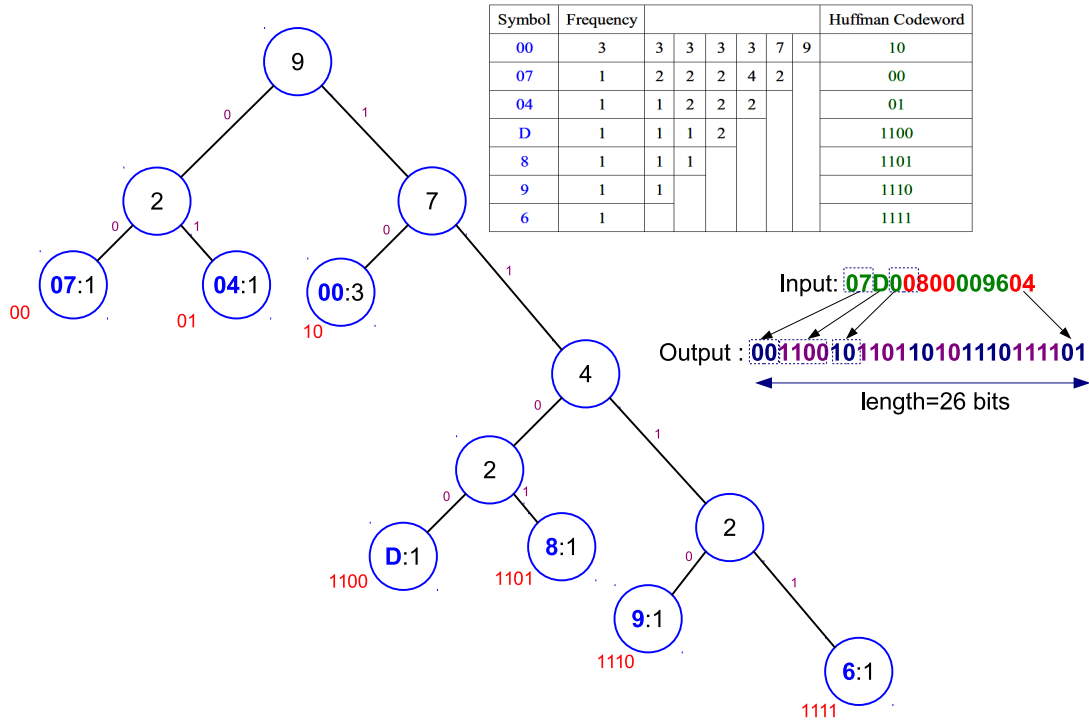


Figure 3.10: Combination of Huffman encoding and Run-Length encoding.

Table 3.4: Information encoding

	Layer ₁	Layer ₂	Layer ₃
Attribute String	0123	4567	89AB
Attribute Positioning	FFF7	F333	F7FF
Hexadecimal Stamp	07D00800009604	0008012	20000500520064
Binary Stamp	00000111110100000000100 00000000000000000100101 1000000100 Length=56	00000000000010000000000 10010 Length=28	00100000000000000000010 10000000001010010000000 0001100100 Length=56
LLH Stamp	01101100001100000010111 1001111 (see Figure3.9) Length=30 Reduction=46%	000100111110 Length=12 Reduction=57%	11000001000101100011101 111 Length=26 Reduction=53%
RLH Stamp	00110010110110101110111 101 (see Figure3.10) Length=26 Reduction=53%	000100111110 Length=12 Reduction=57%	1100010010110011101111 Length=22 Reduction=60%

Table 3.4 provides a comparison between both LLH (our approach) and RLH (i.e., a combination of Huffman and Run-Length algorithms) coding algorithms in terms of reduction of bits in encryption. For this comparison, we use the definition of attributes and their

Table 3.5: Information encoding - Summary

	Layer ₁	Layer ₂	Layer ₃
Hex-smp	07D00800009604	0008012	20000500520064
Bin-smp _{length}	56 bits	28 bits	56 bits
LLH-smp _{length}	30 bits (see Figure 3.9)	12 bits	26 bits
LLH-smp _{reduction}	46%	57%	53%
RLH-smp _{length}	26 bits (see Figure 3.10)	12 bits	22 bits
RLH-smp _{reduction}	53%	57%	60%

corresponding values (for a sample resource), presented in Table 3.3. As it is shown in Table 3.3, the defined attributes are categorized in 3 layers. Accordingly, we create layer-stamp for each layer using the attribute values mentioned in Table 3.3 (sample values) and the given attribute-string for each layer in Table 3.4. Hexadecimal-Stamp (Hex-smp) and Binary-Stamp (Bin-smp) specify the resulted layer-stamps in hexadecimal and binary respectively. We encrypt hexadecimal-stamps using both LLH and RLH coding algorithms. LLH-Stamp (LLH-smp) and RLH-Stamp (RLH-smp) provide the encoded layer-stamps using LLH and RLH respectively.

Table 3.5 summarizes the aforementioned comparison between LLH and RLH. Bin-smp_{length} shows the length of Binary-Stamp (number of bits) for each layer. LLH-smp_{length} and RLH-smp_{length} are also the lengths of each encoded layer-stamp using LLH and RLH encoding accordingly. Similarly, LLH-smp_{reduction} and RLH-smp_{reduction} demonstrate the rate of reduction for encoded layer-stamps for both methods. The rate of reduction is the ratio of bits-reduction (number of bits for input string minus number of bits for output string) to number of bits for input string. As we can see in Table 3.5, RLH provides better rate of reduction in the range of [53%, 60%] while the rate of reduction for LLH is [46%, 53%]. However, the rate of reduction is not the only important performance criteria in deciding to use a coding approach, rather, there are other important aspects. Among them the cost of encoding (in terms of memory) is very important. In fact, an encoding process generally creates two type of information in output, including the encoded information and the information which is required for decoding process (e.g., coding-info-string which stores the list of symbols and their corresponding codes). For an encoding process, the cost of encoding specifies the amount of memory which is required to store encoding information (for the purpose of later decoding).

Table 3.6: Cost of encoding (required space): NOS=Number of Fixed-Length Symbols, BPS=Required Bits per Symbol, BFL=Required Bits for the Fixed-Length, MCL=Maximum Permitted Codeword-Length, MBC=Maximum Required Bits for the Codewords, LID=Length of Sample Input Data by Bits, TCB=Total Cost or Maximum Cost by Bits.

	NOS	BPS	BFL	MCL	MBC	LID	TCB
LLH	$n=16$	4	4	$l=16$	$n * l=256$	any-length	384
RLH	$n * n=256$	8	4	$l=16$	$n * l=4096$	any-length	7168
e.g. layer ₁ LLH	7	4	4	4	23	56	79
e.g. layer ₁ RLH	7	8	4	4	22	56	106

In Table 3.6, we compare RLH to our LLH encoding method in different aspects. As we can see in the table, LLH creates the maximum cost of 384 bits for each layer encoding while its rate of reduction is more than 30%. It means that using this scheme, we would be able to encode unlimited number of attribute-values with the maximum memory cost of 384 bits which is very low-cost. Table 3.6 also shows that despite our LLH algorithm, the maximum memory

cost for RLH is too high (7168 bits per each layer encoding). In overall, RLH provides better rate of reduction (compared to LLH), but it significantly suffers from its large encryption cost.

3.6 Conclusion

In this chapter, we presented a new hierarchical resource description model having this aspect of how the hardware resource information is distributed between computing-enabled peers (resources) and how the hardware resource capabilities can be described and derived from hardware descriptions. The proposed hierarchical hardware modeling and resource description mechanism is able to abstract the characteristics and behavior of the large scale many-core-enabled computing systems in a way that both computational and communicational system properties are well represented to provide a close estimation of the real system while avoiding to describe the hardware at the cycle-accurate level. The depth of hierarchy (i.e. number of layers in resource description model) and the definition of each layer also can range from very high level (e.g. super clusters, clusters) to very low level (e.g. processing core, ALUs) depending on architectural designing aspects. The proposed resource description model is very detailed and general for resource discovery and mapping while dealing with aspects such as capturing static and dynamic capabilities of hardware resources, and making distribution of hardware information scalable.

In comparison to the current literature, our description model/language is a domain specific language with an embedded encryption method, specifically designed for the purpose of resource discovery in large dimension many-core enabled future computing systems with capability for scalable distribution of encrypted resource information across the system. Furthermore, it is highly expressive and flexible to describe various complex queries/systems and detailed attributes/layers, making feasible to provide almost most of the necessary description requirements (scalability, flexibility, expressiveness and compact design) for resource sharing and discovery in such future systems. Furthermore, the proposed resource description model is computing-oriented, which means, unlike most of the approaches in the literature, we describe all types of resources (Compute, Network and Storage) from the viewpoint of computation. This inherently provides capability for the resource description model for being more adapted to the applications like discovering processors in distributed computing systems.

Chapter 4

Hybrid Adaptive Resource Discovery

Resource discovery is one of the most important building block for large scale distributed computing systems. In this chapter we propose a novel resource discovery approach for such systems. Along this line, we also propose a new hierarchical clustering algorithm which automates the process of the optimally composing communicating groups in a dynamic way while preserving the proximity of the nodes. Additionally, we present a new resource management model which can be applied and adjusted to different requirements of future complex computing environments.

4.1 Introduction

Resource discovery aims to match the application's demands to the existing (distributed) resources, by discovering and finding resources at run-time, and then selecting the best resource that matches the application running requirements. This chapter proposes Hybrid Adaptive Resource Discovery (HARD), a novel efficient and highly scalable resource-discovery approach which is built upon a virtual hierarchical overlay based on self-organization and self-adaptation of processing resources in the system where the computing resources are organized into distributed hierarchies according to a proposed hierarchical resource description model (i.e., multi-layered resource description). The proposed approach supports distributed query processing, and aggregation of discovery results within and across hierarchical-layers, by leveraging various distributed resource discovery services and functionalities in the system, implemented using different adapted algorithms and mechanisms in each level of the hierarchy.

Operationally, each layer consists of a peer-to-peer architecture of modules that, by interacting with each other, provide a global view of the resource availability in a large, dynamic and heterogeneous distributed environment. The proposed resource discovery model provides the adaptability and flexibility to perform complex querying by supporting a large set of significant querying features (such as multi-dimensional, range and aggregate querying) while supporting exact and partial matching, both for static and dynamic object contents.

4.2 Requirements and Design Principles

In large scale distributed computing systems such as Cloud, Grid and High Performance Computing (HPC) supercomputers, it is not feasible for the control system to have complete and perfect knowledge of the entire system due to the magnitude and diversity of the amount of resources (e.g., processors). This is where the role of resource discovery comes into play and becomes very significant to provide on-demand information about the system resources. For uniformity of discussion, for such computing environments we will address all kind of control systems as Distributed Operation Systems (DOSs), although their realization may be quite different (depending on the specific large scale computing technology under discussion).

The general architecture design of a large scale distributed computing environment can be illustrated as a set of distributed hierarchies like the one shown in the Figure 2.5. Depending on the level of hierarchies in the architecture and the other designing aspects of a DOS, we may define control entities in different levels. For instance, as it is discussed in [4], we can describe the entities of Main-Control (i.e., DOS main-kernel), Micro-Control (i.e., DOS micro-kernel) and Nano-Control (i.e., DOS nano-kernel), which are providing either maximal, moderate or minimal amount of capabilities, functionalities and services in the system. These control entities may differ in terms of service types which they can dynamically instantiate on demand. The instances of these entities are positioned in the system in a way to map the structure of the underlying distributed hierarchies (e.g., deploying the main-control instances in the hierarchies head-nodes and the nano-control instances in the leaf-nodes).

4.2.1 Assumptions and Definitions

HARD is based on methods and techniques to distribute resource information, update and exchange resource data and query and search space exploration, specially when considering the particular challenges and requirements of distributed operating systems. In order to create such solution, we impose the following assumptions on the design process:

- We assume that each single resource (i.e., single core) has its own unique ID (e.g., IP, network-ID, Chip-ID, processor-ID) which can specify its address in the entire system. Addressing information can potentially be provided by the operating system or other system component, and is out of scope in our research work.
- We assume that the target computing environment can behave like an unstructured Peer-to-Peer (P2P) distributed environment containing large number of peers (i.e., resource or computing entities) where each peer only knows about itself and its connection gates.
- We assume that the heterogeneity of the resources is very high. i.e., high heterogeneity in terms of computation and communication characteristics, such as heterogeneity of the processing resources (e.g., Central Processing Unit (CPU), Graphics Processing Unit (GPU), etc.), interconnecting and communication networks, memory hierarchies, etc.
- We assume that the computing environment is highly dynamic, so that no static configuration would be possible.

The proposed discovery approach is based on a hybrid virtual overlay network, which will be constructed automatically over the underlying physical network. This virtual overlay contains virtual-nodes that are organized in distributed hierarchies. In this work, we present a

3-layered instantiation of HARD (HARD3), which proposes three levels of resource discovery services (i.e., three types of resource discovery components): Resource Requester (RR), Resource Information Provider of type Query Management Service (RP-QMS) and Resource Information Provider of type Super Query Management Service (RP-SQMS). We will discuss these services in more detail later in this chapter (see Sections 4.2.3 and 4.5). We must note that, in this thesis, we use the term “HARD3” interchangeably with the term “HARD” depending on which aspect we want to emphasize. Both terms are used to specify our proposed approach for resource discovery. However, the term “HARD” is a more general term than the term “HARD3” (i.e., HARD3 refers to a specific implementation of HARD). We will also use the following definitions:

- We use the notion of “network node” to describe a group of processors that share a common network interface (e.g., they have a common IP address to communicate with other network nodes).
- A node-stamp is defined as a fingerprint of all predefined characteristics of a node in a specific layer.
- We define the notion of a “virtual node (vnode)” as a group of homogeneous resources, which are not necessarily positioned on a common physical node (e.g., a CPU). Rather they are positioned within a common vicinity that is described by parameters such as number of hops or interconnect latency (i.e., resources are grouped within vnodes by proximity and similarity). We use the generic notion of “node” instead of “vnode”, which has the same meaning with more emphasizing on the positioning of the “vnode” in the underlying self-organized virtual overlay. In addition, in this thesis, the term “Source of Resource (SoR)” or source of resources might be used instead of vnode which also has the same meaning with more emphasizing on the resource-supply-quality aspects such as SoR’s stability or SoR’s strength.
- Every vnode is automatically assigned a module-role (i.e., *vnode*-type) in a self-organized and distributed fashion. The module-role defines the specific *vnode*’s role to play in the overall distributed resource discovery operations. We discuss module-roles with more detail in Section 4.2.3.
- The term “cell” is used to denote a group of vnodes which are sharing a common SQMS-ID and the term “mini-cell” is used to refer a group of vnodes with a common QMS-ID.
- For better illustration and evaluation of our approach, we use throughout this thesis the notion of “resource” to refer to “computational resource” (i.e., physical processing cores) and we discuss resource discovery mostly from the point of view of computational capacity. Nevertheless, HARD is fully generic, and applicable to other types of resources (e.g., storage and networking resources).

HARD3 is implemented as part of the Service oriented Operating System (S(o)OS) concept [4], which in itself is an example of a DOS. Each DOS kernel, regardless of its level (i.e Main, Micro, Nano, etc.), provides support for a single RR service. Furthermore, the kernels depending on the level which are positioned in the hierarchy provide support for other types of resource discovery services such as RP-QMS and RP-SQMS (e.g., main-kernels or

micro-kernels may provide RP-SQMS or RP-QMS services). We must note that it may be possible that a kernel simultaneously provides either one, two or all of these discovery services. For example, the kernels in the top level of hierarchy support all kind of discovery services.

4.2.2 Resource Description

We leverage the resource description model presented in Chapter 3 to represent information extracted from underlying hardware infrastructures as well as the query description (i.e., specifying required/desired resources for each discovery request).

4.2.3 System Architecture

Modern and future distributed computing systems (e.g., Jungle Computing Systems (JCSs) [1, 401–405]) are highly-hierarchical and highly-heterogeneous in nature. Accordingly, the HARD architecture deploys various layer-based hybrid adaptive mechanisms (i.e., inter-layers and intra-layers methods) in order to efficiently direct discovery requests to the proper resources across layers. This means that, according to properties and characteristics of each layer in the hierarchy, HARD proposes a set of specific adapted methods, which have been designed to obtain the maximum discovery efficiency on the target layer, while an integrated and coherent approach is used to traverse layers in hierarchy. Figure 4.1 depicts the overall architecture of HARD3, highlighting users, main services, underlying techniques and organization of computing resources in different layers.

We build our system architecture based on a self organized virtual hybrid overlay. In order to create the virtual overlay, at first, the unstructured resources in the system are organized within vnodes according to their homogeneity and proximity parameters (i.e., their similarities and locations). In the next step vnodes start to negotiate with each other in a multi-round distributed fashion to seek agreement on the contribution (i.e., module-role or vnode type) of each party in the overlay hierarchy. As negotiations evolve, each vnode shapes its own system-view by improving and consolidating its own knowledge on the entire system. The resulting overlay contains three different types of virtual-nodes: Leaf-Nodes (LNs), Aggregate-Nodes (ANs) and Super-Nodes (SNs) which take position in $layer_{ln}$, $layer_{an}$ and $layer_{sn}$ of the hierarchy respectively. Depending on the vnode type (i.e., module-role), each virtual-node provides different HARD3 services (e.g., RP-QMS and RP-SQMS). vnodes in the upper layers are able to provide discovery services specific to their own layer and all the services in the lower layers. vnodes respond to the discovery demands based on their module-roles as well as the immediate requirements of the triggered communication events. For example, a vnode in $layer_{sn}$ (i.e., a super-node) provides RP-SQMS service. However, depending on the properties of the received communication events, it may also provide RP-QMS or RR services or participate in the overall discovery procedure by playing a role of a leaf-node.

As it is represented in Figure 4.1, the leaf-nodes in $layer_{ln}$ are organized in Distributed Hash Tables (DHTs) based on the multidimensional fingerprint (layer-stamp) of each participating vnode. In fact, the leaf-nodes participate in a core-level specification-based DHT ring where the sibling nodes (i.e., the vnodes with similar resources) are linearly organized in linked lists with single entries on the DHT ring. Similarly, aggregate-nodes (in $layer_{an}$) and super-nodes (in $layer_{sn}$) regardless of their module-role, participate in DHT. For each group of LNs which elect a single AN/SN as their common resource provider (RP-QMS or RP-SQMS), the DHT is composed of all the vnodes in the group (containing LN members and either a single AN or

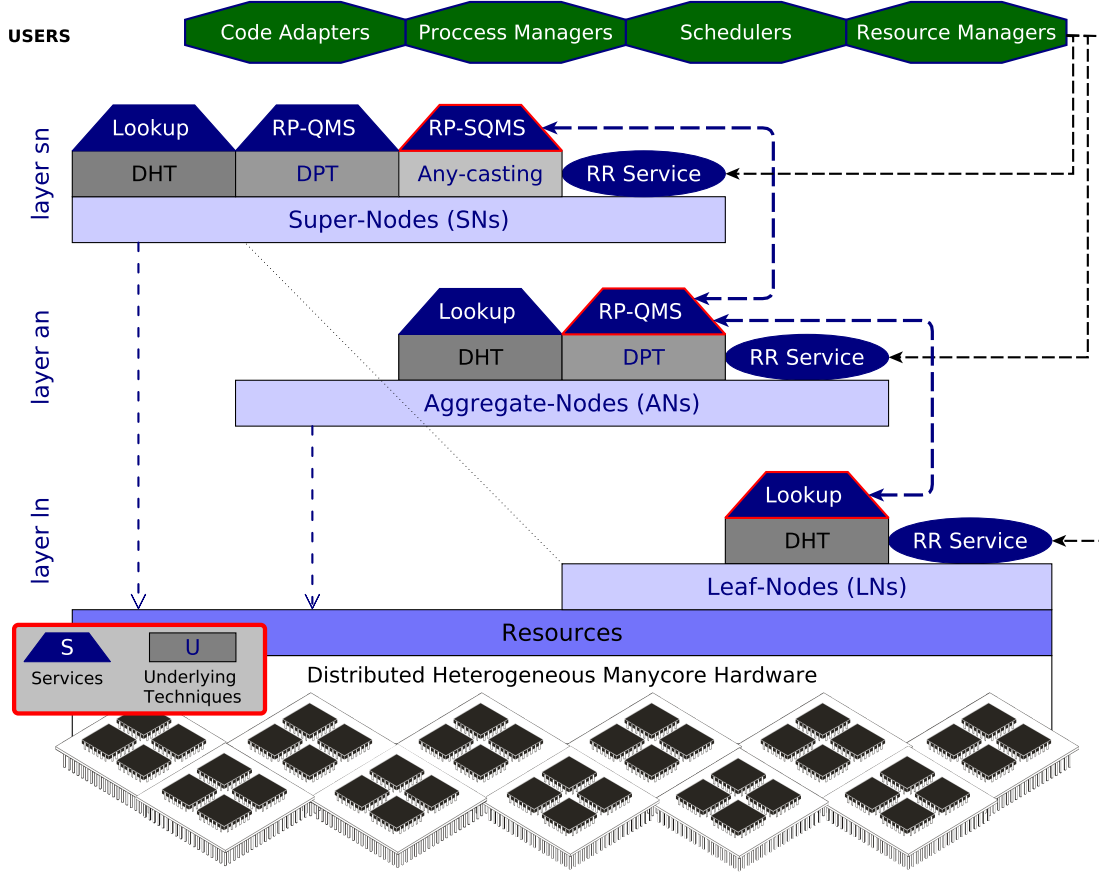


Figure 4.1: HARD3 overall system architecture.

a single SN). In other words, all vnodes, regardless of their module-role, are able to perform **Lookup** queries over DHTs (refer to section 4.6.1 for a detailed description of the proposed algorithm for DHT lookup).

The reason to use DHT in the core-level (*layer_{ln}*) of our architecture is due to the following aspects: (a) DHT is scalable: this is specially important when considering the potentially large number of cores that can reside on a single die (in future systems). (b) DHT is fast, reliable, fault tolerant and deterministic: resource discovery in the core-level is much more sensitive to speed than querying in the network-level, due to the tightly coupled design of many-core processors. In many-core level, resource discovery might be ineffective if it fails to provide required information in an adequate amount of time (e.g., discovery latency might have a direct impact on the cost of execution migration in a many-core environment). Moreover, in such highly sensitive environments, it is essential for a discovery method to operate reliably and provides deterministic results (undetermined results might have cost by reprocessing the query or exploring an already visited search space). (c) DHT maintenance is low cost (in terms of memory and communication): a very small finger-table is required to be maintained in each vnode in *layer_{sn}*. (d) DHT supports attribute-based query description: this makes DHTs more compatible to our attribute-based resource description model. On the other hand, DHTs originally do not support semantic-based querying. To solve this issue, in our DHT

variation, we enhanced the original Chord DHT to support a similarity algorithm which makes feasible similar-matching instead of exact-matching (HARD supports both modes of matching through specifying the desired matching mode in the query by the user).

Query Management Service (QMS) is a service which provides query processing facilities in layer_{an} . It uses a probability-based mechanism to guide queries among a group of aggregate-nodes which share a single super-node as the resource provider (i.e., RP-SQMS). During the discovery procedure, Distributed Probability Tables (DPTs) cooperate with each other in a set of dynamic distributed learning processes, which are adapted to the progressive environmental changes. For each AN, its local probability table dynamically collects, aggregates and updates information about the status of the overall resources in the system, gathered from all transacted queries and results through the AN itself. By using this DPT technique, the network that connects ANs becomes increasingly resource-aware, as the number of traversed queries increases across the system (refer to section 4.6.3 for detailed description of the proposed methods for DPTs and querying in layer_{an}).

We use DPT as a base method in the die-level (layer_{an}) due to its scalability, dynamicity, efficiency and also its compact structure. Comparing to DHT, DPT provides probabilistic results instead of deterministic results. But this not a drawback, since DPTs operate in the middle-level of HARD architecture which does not need to provide deterministic results. The reason is that, queries are not going to be concluded in layer_{an} . In fact, a query processing starts from the top-level (layer_{sn}) (of course, if there exist any query conditions for this layer) and then goes to the middle-level (layer_{an}) and finally it could be concluded in the lower-level (layer_{ln}). Furthermore, we enhance our DPT approach by introducing a SoR mechanism which can help DPT to provide deterministic results whenever feasible.

The compact design and dynamic nature of DPT provides a facility to efficiently cope with dynamic changes in the environment (e.g., unavailability of resources due to resource failure, resource reservation, etc). Each $vnode$ in layer_{an} maintains a small probability table. Depending on the number of predefined attributes in the system, a property table may include multiple records (called resource-type or resource-category records), where each record represents the aggregated probability information for all neighbors with respect to the overall query transaction data (monitoring data), collected and analyzed, for a single attribute over a predefined specific range of values. In fact, each resource-type record includes probability factors for all neighbors as well as a suggestion of a SoR (a $vnode$ which deterministically can provide resources, matched with the resource-type definition of the record). We also note that probability tables only cover attributes defined for layer_{ln} and layer_{an} . We discuss further details of DPT and SoR mechanisms in Section 4.6.3.

Super Query Management Service (SQMS) is a specific QMS which provides additional capabilities to support query forwarding in layer_{sn} . For instance, as we can see in Figure 4.1, super-nodes (i.e., SQMS providers) are able to concurrently provide multiple services (such as lookup, QMS, SQMS and RR) for the different triggered communication events (refer to section 4.5.5). The query forwarding in layer_{sn} uses the specifications of the resources in the node-level to conduct a specification based anycasting method to direct the queries among SNs. It uses the top level layer-stamps to create anycast groups, while nodes in this layer are able to automatically adjust to the anycast group they are interested in based on their specifications in layer_{sn} (refer to section 4.6.5). We use anycasting as a base method for querying in the network-level (layer_{sn}) due to its scalability, efficiency and its powerful features which make it more adequate and compatible for resource discovery in computing systems with a large number of network connected nodes (we discuss this further in Section 4.6.4).

Depending on the specific DOS architecture, HARD3 users could be resource management entities, schedulers, process managers or even code adapters. Each user would be able to perform resource discovery through invocation of a RR entity. RR in turn sends the given query to its local RP-QMS. Due to the type of query and the user's demand (e.g., simple single resource, multiple heterogeneous resources, complex resource graph containing the constraints for inter resource communications, etc.) RP-QMS splits the main-query to a set of sub-queries and chooses the appropriate layer that each sub-query must start to process. Finally, the RP-QMS that originated the sub-queries, aggregates the discovery results, and responds the RR with a set of resource matches that optimally satisfy the main-query's demand. In the remaining of this chapter we elaborate more on the mechanisms proposed above.

4.3 Self-organization and Self-configuration

In large-scale service-oriented systems, it is desirable for services to be able to collaborate with each other in order to achieve a common goal without relying on any centralized control (e.g., fully centralized or hierarchical architectures). Such systems are subject to dynamicity and scalability due to bottlenecks and single point of failures. But employing a fully-distributed system has its drawbacks too. Furthermore, the service distribution and allocation among a large set of dynamic resources would become too complex to be managed by using manual or statical configuration. Services need to become self- adaptive to maintain the entire system performance and functionality. Along this line, it is necessary to employ an efficient self-deployment and self-configuration mechanism which dynamically creates communicating groups of resources by building logical overlays on top of network topology. In this section we present a multi-step, load-balanced, self-organized, clustering algorithm for overlay establishment (particularly for resource discovery) in distributed environments, which preserves the locality of the nodes within the groups while it deals with efficiency and scalability.

4.3.1 Description of Algorithm

This section presents our algorithm to construct a three-layer overlay, including LN, AN and SN layers, as we mentioned before, in Section 4.2.3. Using this way, the resulted overlay can support query processing for the resource discovery, through dynamic self-organization of resources into groups and distributed hierarchies (including dynamic mechanisms for new group creation and group maintenance). Figure 4.2 shows examples of cluster shapes (for two systems with different sizes and random network topologies) before and after applying the proposed self-organization algorithm. The module-roles for all nodes are identified based on the node-color (grey: SNs; red: ANs; white: nodes that their module-roles are not specified yet; other colors: LNs; nodes with the same color are members of a common group). Our algorithm contains multiple steps to establish and create a dynamic hierarchical overlay on top of the network topology. Here, we explain these steps with regard to the general assumptions mentioned in Section 4.2.1. But before that, we define a set of terms (i.e., input parameters and criterias) which are used to structure our algorithm and they can be additionally used to evaluate the algorithm's efficiency. These terms are as follows:

1. Group Diameter (G_d): It is the maximum number of hops between an aggregate node of a group and other leaf-node members within the group. (i.e., it is the maximum distance between an aggregate-node and a leaf-node within a group).

2. Locality Factor (k): It is a defined criteria to indicate the proximity of the nodes within a group. $k = \frac{1}{G_d}$, where k is the locality factor and G_d is the group diameter.
3. Neighboring Indicator (N_i): It clarifies the definition of the neighboring nodes through specifying the maximum number of hops between the source node and the potential neighbors.
4. Grouping Variables: They refer to the overlay design parameters (i.e., the validated range for the group's size in different layers of hierarchy) such as Maximum/Minimum number of allowed nodes per group in AN/SN layer.

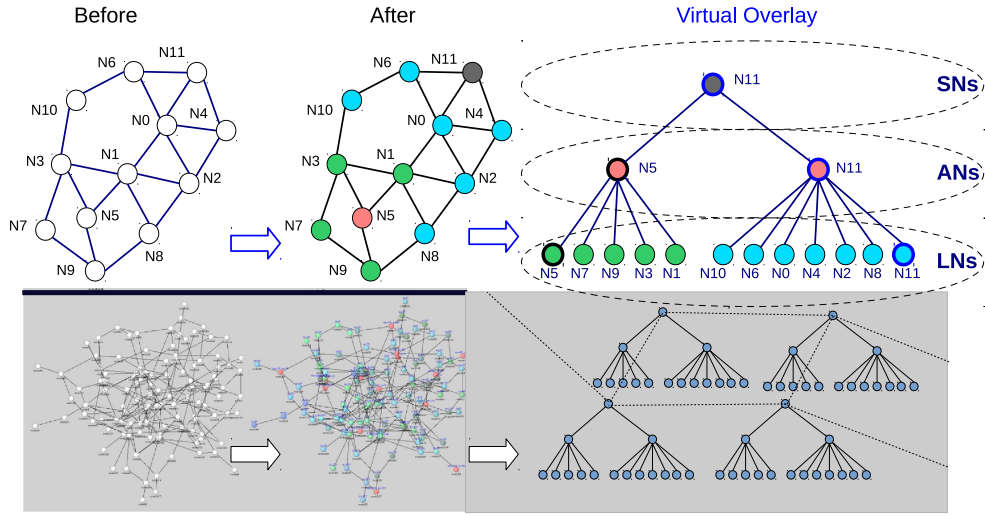


Figure 4.2: Examples of cluster shapes (for two systems with different sizes and random network topologies) before and after applying the self-organization algorithm.

The following gives an overall description of the algorithm before going into the details. On the first step of the algorithm, each node discovers the underlying topology by recognizing its direct set of connecting neighbors. Afterwards, on the second step, nodes start to negotiate and exchange messages with each other in order to efficiently create the first level of the communicating groups which consists of specifying a group and a role (either leaf-node or aggregate-node) within the group for every one of the nodes in the system. On the next step, an optimization algorithm is applied to enhance the results of the previous step by merging the groups that have their sizes below the required threshold. Finally, on the last step, the aggregate-nodes of each group must participate in a process to elect the super-nodes which leads to establishing the second level of the communicating groups. Table 4.1 provides a summary of the types of communication messages that are used for clustering and are used for overlay construction. In the remaining of this section we discuss the aforementioned steps in more detail.

Step 1 (discovering the underlying topology): A node initializes the procedure by sending **ID-Request** messages to all of its local connecting gates in order to obtain information about the possible neighbors. By default, the module-roles and the grouping variables (such as resource-id, QMS-ID and SQMS-ID) of all the nodes are unknown. The resource-id denotes the

Table 4.1: Summary of the communication events for the overlay construction.

Event	Description
ID-Request	Asking for certain information about the other-side of a direct connection link
ID-Reply	Returning the required information specified by ID-Request to the requester
WakeUp	Initializing an assigned procedure of a particular type for a node
Join-Request	Asking a node to establish a new group or to add a new member to an existed group
Join-Accept	Informing a node that its Join-Request has been accepted under certain conditions and the grouping direction is from sender(an aggregate-node) to receiver(a leaf-node)
Update	Informing a node that its Join-Request has been accepted under certain conditions and the grouping direction is from receiver(an aggregate-node) to sender(a leaf-node)
Change	Informing a node that it needs to change its grouping information to be adapted into the new group
Failed-Group	Making a node unreserved (free) by changing its status to unknown
Grouping-Info	Informing a group member about its group information
Group-Checking	A spot node (a node which has received the Grouping-Info from at least two different groups) will detect and analyze all the group-merging possibilities (i.e., creating merging-list)
Start-Merging	A Group Representative (GR) receives a group merging proposal. i.e., A GR node receives a suggestion to merge with a specific larger validated group while the locality preservation and the group making feasibility are guaranteed
Merging-Request	Asking a group to be merged with the requester group
Super-Node-Election	Asking a node to participate in the Super-Node-Election by voting to a certain node

node's address (e.g., IP address) while the QMS-ID (aggregate-node's address) and SQMS-ID (super-node's address) specify the layer and the group that the node belongs to. Upon receiving **ID-Request** message at a destination, the receiving node records the sender information and sends its information to the requester through a message of type **ID-Reply**. Meanwhile, it checks its local information about the other neighbors and, if there is still some missing information, it propagates the **ID-Request** messages to all of its connecting gates except the one that already knows about the other side of the connecting edge (see Figure 4.3).

Step 2 (creating the first level of the communicating groups): Each node by itself generates a random delay bounded in a specific time frame. Consequently, a **WakeUp** event is automatically triggered when the delay is over. Upon the occurrence of the event, the node checks its state and if the value of the module-role is still unknown it sends a message of type **Join-Request** to all its neighbors (see Algorithm 1). The information about the neighbors was already collected in step 1. Depending on the state of the destination nodes, arrival of the **Join-Request** messages cause different reactions in the receiver nodes. When the module-role of the receiver node is unknown, it changes the current state of its module-role to LN, while it also sets the value of its grouping variable for the first layer (i.e., QMS-ID) according to the address of the request sender. In addition to that, it sends a message of type **Join-Accept** to the requester address. It means that the receiver node is ready to join a group, which can potentially be created by the requester itself as the aggregate-node. In the case that the module-role state of a receiver node is LN, which means that this node already belongs to a group, it relays the request message to its assigned aggregate-node by forwarding the message to the address mentioned in the QMS-ID. Another possibility is that the receiver node of a **Join-Request** message is an aggregate-node (i.e., the node already belongs to a group and its module-role is AN). Each aggregate-node locally registers its LN members. A group of one AN and its LN members is called an AN-Group. Although, the number of LN members in a group is restricted to the maximum and minimum allowed size of the AN-Groups, it has to be clarified by the values of the overlay design input parameters. Thus, for each new joining request, the aggregate-node (as the receiver of a **Join-Request** message) must examine

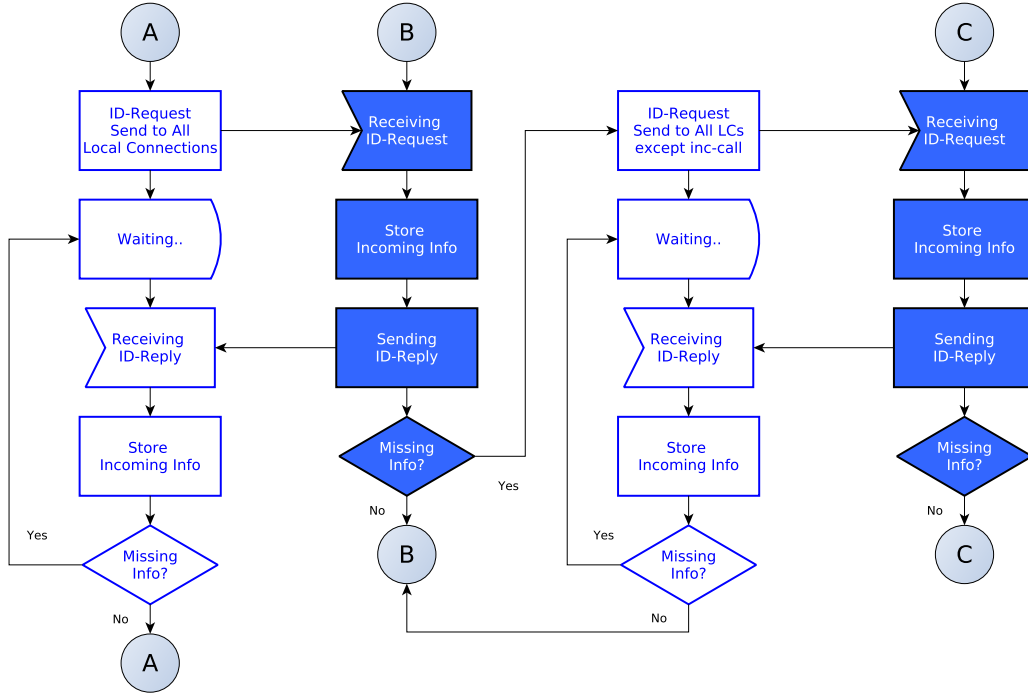


Figure 4.3: Overlay construction : Stage 1.

the possibility of adding the new member to the group and if it is feasible it must send an **Update** message to the original requester containing the information of the aggregate node saying that the requester is allowed to join the group. Otherwise the request message (i.e., the **Join-Request** message) is just ignored. However, if the receiver node decides to ignore the request message, it caches the requester information as a potential remote aggregate-node for the purpose of super-node election on the later steps of the algorithm (see Algorithm 1). Figure 4.4 demonstrates different situations of message processing for overlay making (with maximum group size=3). These situations include: A) On a **WakeUp** event, a node has sent a **JoinRequest** (or **JRequest**) to all of its neighbors, B) On a **JRequest** event, a node has accepted the request, C) On a **Join-Accept** (or **JAccept**) event, a two-members group has created, D) On a **JAccept** event, a reserved (devoted) node has failed to become a group member, E) On a **FailedGroup** (**FGroup**) event, a reserved node has released, F) On a **JRequest** event, a **JRequest** has forwarded to a remote-AN, G) On a **JRequest** event, the remote-AN has accepted the request and has tried to update the requester, H) On an **Update** event, an existing group has denied to become a part of another group and finally in I) On a **WakeUp** event, a non-grouped node has waked up and sent a **JRequest** to its neighbors.

Generally, the most sensible reactions of the nodes to have a **Join-Request** message are sending either **Join-Accept** or **Update** messages, which specify the direction of the grouping from sender to the receiver or vice-versa. When a node receives a **Join-Accept**, it means that the node is already qualified by at least one node to establish a new group as an aggregate-node. However it might be possible that the node's state has changed during the time between issuing the **Join-Request** by the node and getting the **Join-Accept** from a particular node. Therefore the node behaves according to its current state to handle the incoming **Join-Accept**.

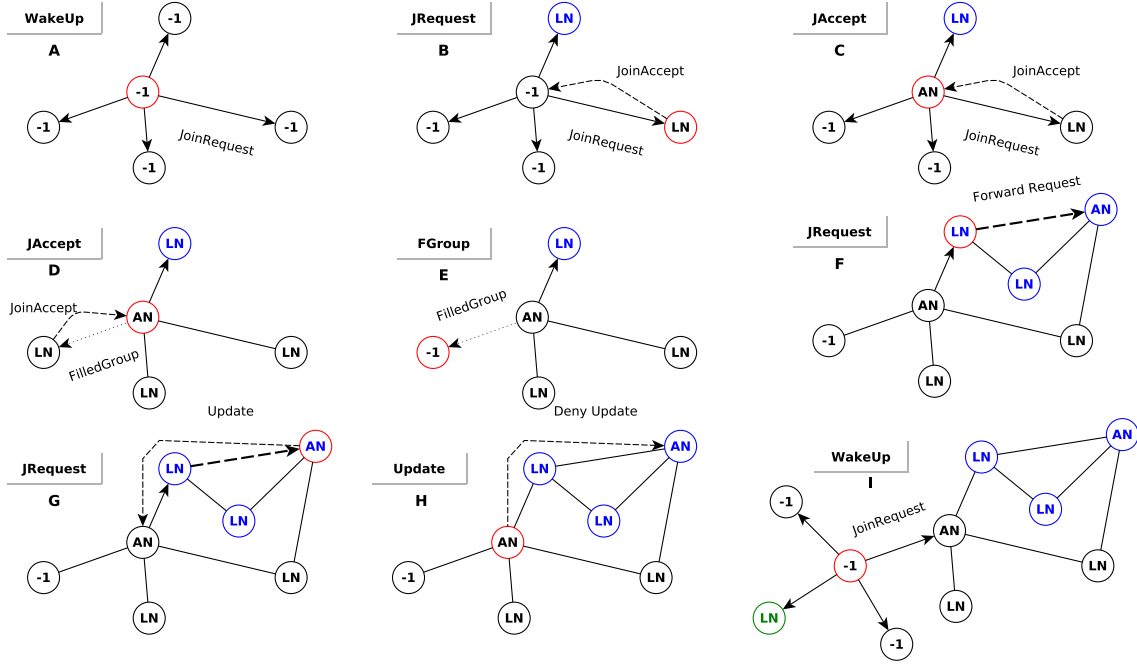


Figure 4.4: Examples of various situations of message processing for overlay making (with maximum group size=3)

Algorithm 1: WakeUp,JRequest.

```

Generate(WakeUp,RandomDelay)
On-WakeUp:
if local – node.state is unknown then
  for all remote – node IN neighbours – set( $N_i$ ) do
    sendMsg(JRequest,remote – node)
  end for
end if
On-JRequest:
switch (local – node.state)
case unknown:
  set local – node.state as LN
  set local – node.AN as msg.sender
  sendMsg(JAccept,msg.sender)
case LN:
  forward(msg,local – node.AN)
default:
  if checkGP(GVars,k) > 0 then
    sendMsg(Update,msg.sender)
    AMembers.add(msg.sender)
  else
    Cache(msg.sender.info)
  end if
end switch

```

If the current module-role is unknown, it changes its state to AN, and a new group is created with at least two members (including the current node) and if its current state is LN, it simply forwards the **Join-Accept** message to its aggregate-node, which is responsible for making decisions about the joining possibility of the new members. We must notice that this would

happen only if the issuer of the **Join-Accept** is a node different from the local aggregate-node. In other cases, when the receiver of the **Join-Accept** is an aggregate-node, according to the group making policies, the AN receiver decides to add the **Join-Accept** issuer to its local group. It checks if the message has been relayed from one of the current leaf-node members and if so, it sends a message of type **Change** to the original sender to update its grouping information for the new aggregate-node. Otherwise, the aggregate-node just records the information of the new member for the later query processing (e.g., resource discovery query). In other situations if the AN (as the receiver of a **Join-Accept** message) refuses admitting the potential new member (i.e. the sender which has already devoted itself for being a member of the AN receiver) to the group (based on the policies like group size or locality factor) it sends a **Failed-Group** message to the sender to make the sender node free by changing its status to unknown. In fact, when a node issues a **Join-Accept**, that node will be temporarily reserved for a potential grouping led by a particular aggregate-node. A **Failed-Group** message releases the reserved node (see Algorithm 2).

Algorithm 2: JAccept.

```

On-JAccept:
switch (local – node.state)
case unknown:
    set local – node.state as AN
    set local – node.AN as local – node
    AMembers.add(msg.sender)
case LN:
    if msg.sender is local – node.AN then
        set local – node.state as AN
        set local – node.AN as local – node
    else
        msg.flag.set(FJAccept)
        forward(msg, local – node.AN)
    end if
default:
    if checkGP(GVars, k) > 0 then
        if msg.flag.get is FJAccept then
            sendMsg(Change, msg.sender)
        end if
    else
        sendMsg(Change, msg.sender)
        Cache(msg.sender.info)
    end if
end switch

```

When a node receives an **Update** message, it means that there is an opportunity for the receiver node to join a group organized by the sender as the aggregate node. So, if the receiver's state is unknown, it sets its module-role as LN and updates its grouping variable (QMS-ID) to the original sender's address. Otherwise, it sends a **DenyUpdate** message to the sender, notifying the remote AN that the current node is not available as it belongs to a different group. The **DenyUpdate** notification will erase the sender node from the list of AN-Group members of the destination node (remote AN). Furthermore, the **Update** message receiver caches the information of the sender (remote AN) when its current state is AN. This information will be used during the group optimization and super-node election processes (see Figure 4.5).

Step 3 (group optimization and merging): The second step will be completed by triggering all the assigned **WakeUp** events in the different individual nodes within various random intervals. Therefore, upon completing the second step, we expect that all the nodes in the system

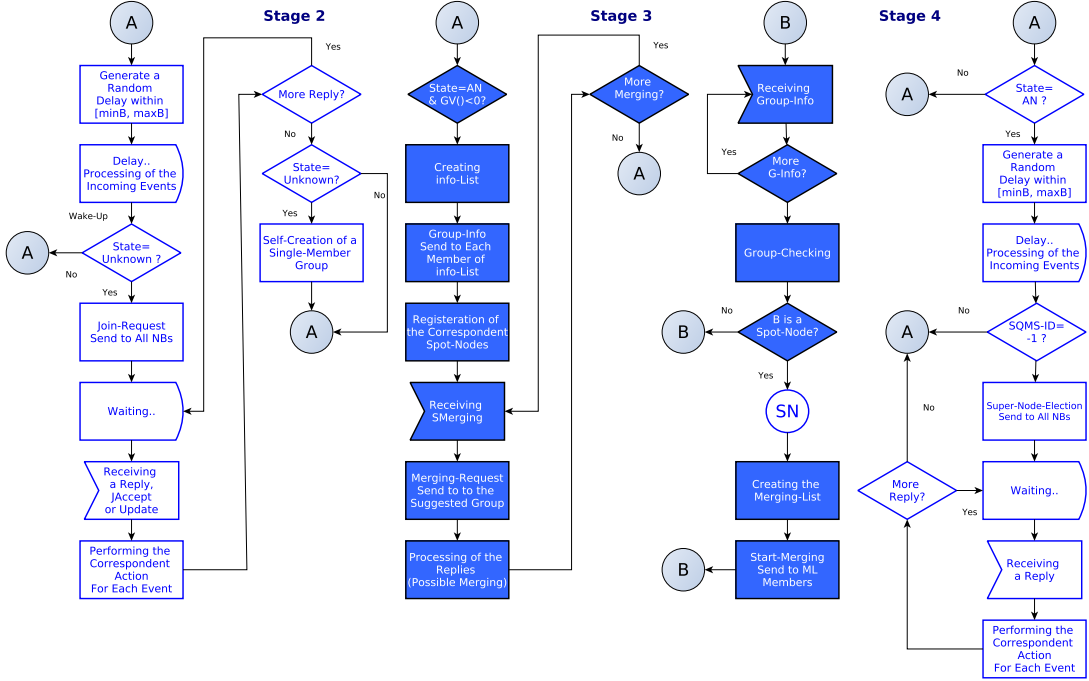


Figure 4.5: Overlay construction considering Stage 2 to Stage 4.

contribute to make a group and find out about their role within the group. But it might still be possible that some nodes are not within the group. In such a situation these nodes will establish their own group as an aggregate-node. They are clustered within a group that could be undersized. Besides, even the groups which are created during the second step may suffer from this problem. For example, the size of a group might be too small compared to the minimum allowed size of a group mentioned as the overlay design parameter. The third step of the overlay creation process tries to solve the aforementioned problem by implementing a grouping optimization algorithm, through merging the small groups in order to create larger groups, which could satisfy the required conditions of the overlay. The group size and the locality-factor are two important parameters for efficient clustering. The locality-factor that is proportional to the group's diameter is evaluated and determined during the second step, while a new member is added to the group. So the grouping mechanism in the second step guaranties that the proximity of the nodes within a group will be preserved. On this step, the aggregate-nodes, as the representatives of other group members, inspect their own grouping conditions and if they notice that their groupings are not efficient particularly in terms of the group size they start to propagate **Grouping-Info** messages to a set of nodes, which would be equal to the union of their members set, their proximity set (neighbors set) and their local set of possible aggregate nodes (cached information). We must note that the neighboring-indicator is an important parameter, which specifies the neighbor's order. For example, if the value of the neighboring-indicator parameter is 1, the proximity set (i.e., neighboring set) only includes the direct neighbors, while for neighboring-indicator=2 the proximity set contains the combination of the direct neighbors and the second level neighbors.

Furthermore, each aggregate node generates and assigns a **Group-Checking** event, which

Algorithm 3: Merging optimization.

```
if .state == AN and GValid(node, GVars, k) < 0 then
  iList = AMembers  $\cup$  CacheANs  $\cup$  Nset( $N_i$ )
  for all remote – node IN iList do
    sendMsg(GInfo, remote – node)
  end for
end if
On-Group Checking:
if ANOps.size > 1 and ANOps.find(.AN) > 0 then
  finalSize = .group.size
  candidateANs.add(.AN)
  ANOps.erase(.AN)
  while finalSize <= Max and ANOps.size! = 0 do
    largestGroup = findLargestGp(ANOps)
    if finalSize + largestGroup.size <= Max then
      finalSize = finalSize + largestGroup.size
      candidateANs.add(largestGroup.AN)
    end if
    ANOps.erase(largestGroup.AN)
  end while
  if candidateANs.size > 1 then
    largestCGp = findLargestGp(candidateANs)
    candidateANs.erase(largestCGp.AN)
    for all remoteAN IN candidateANs do
      sendMsg(SMerging(largestCGp), remoteAN)
    end for
  end if
end if
end if
```

is used to determine the best possible group merging options. **Grouping-Info** messages contain information about the groups such as group size and QMS-ID (the aggregate-node address). The **Grouping-Info** receivers will collect this information for later processing during **Group-Checking** event. When the **Group-Checking** event is triggered, the nodes that have received the **Grouping-Info** from at least two different aggregate-nodes will be distinguished as spot nodes. According to the information collected about the groups in the vicinity, spot nodes detect and analyze all the group-merging possibilities. It must also be taken into account that the final group (after the merge) can't be over-sized and it should be smaller than the maximum allowed value for the overlay design while preserving the locality. On the other hand the merging cost (particularly in terms of communication) to switch the nodes from a small group to a larger group is much lower than switching from a larger group to a small group. The reason is that, during the process of group merging, the aggregate-node of the source group (a group which its members must be merged into the destination group) have to undertake most of responsibility for merging than the aggregate-node of the destination group (a group that has agreed to offer membership to all members of the source group). For the source group, apart from the communication between aggregate-nodes of source and destination groups (to get merging permission from the destination), the responsibility includes sending messages from the aggregate-node of the group to all members (one message to all members), requesting them to update their grouping information through introducing a new aggregate-node. But, the aggregate-node of the destination group is only responsible for making a decision (on either accepting or rejecting the merging request) and informing the requester. This means that the communication cost (in terms of number of transacted messages) for the large-size source groups is much higher than the small-size source groups. Furthermore, in order to merge a source group into a destination group, the aggregate-node of the source group must resign its position, as the group representative, through transferring the information of its

registered LN members to the aggregate-node of the destination group and also changing its module-role from LN to AN. Thus, using small-size source groups (for merge) can be more efficient (in terms of the amount of data transferred) than using large-size source groups, since small-size source groups generate less LN-registration information on their aggregate-nodes.

Due to the aforementioned reasons, the algorithm first chooses the largest validated group among the list and afterwards it selects a set of smaller groups (merging-list) to merge within the largest one. This operation will be iterated for the remaining groups in the list. Finally the spot nodes offer their merging proposals to the related aggregate-nodes through sending **Start-Merging** messages to the aggregate-nodes of the groups within merging list. **Start-Merging** provides a suggestion to the receiver to merge with a specific larger validated group while the locality preservation and the group making feasibility are guaranteed (See Algorithm 3). Consequently the **Start-Merging** receivers send their **Merging-Request** to the proposed destinations and later, upon acceptance of the merging requests by target groups, the requesters will notify their members to upgrade their grouping information according to the new aggregate-node (see Figure 4.5).

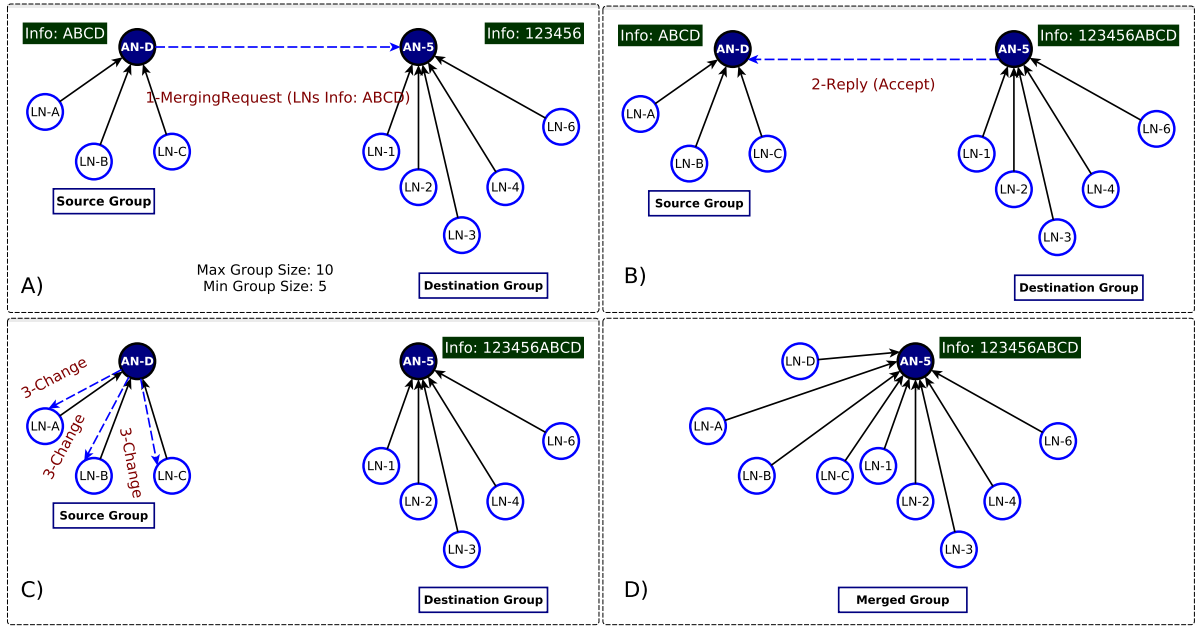


Figure 4.6: An example of merging a source group into a destination group, suggested by a spot node

Figure 4.6 provides an example of group-merging, suggested by a spot node. The merge process includes the following actions: A) AN-D, the aggregate-node of the source group, receives a merging proposal, from a spot node, through a **Start-Merging** message. AN-D sends a **Merging-Request** message (including its members information) to AN-5 (the aggregate-node of the destination group, suggested by the spot node). B) AN-5 receives and accepts the request, and sends a reply to the requester. AN-5 also updates its members information by extracting the new members information from the merge request received. C) After receiving a confirmation from the destination group, AN-D sends a **Change** request to its members,

introducing the new representative of the group which is AN-5. AN-D also changes its module role to LN and becomes LN-D. D) The merge is completed, all members of the source group have merged into the destination group.

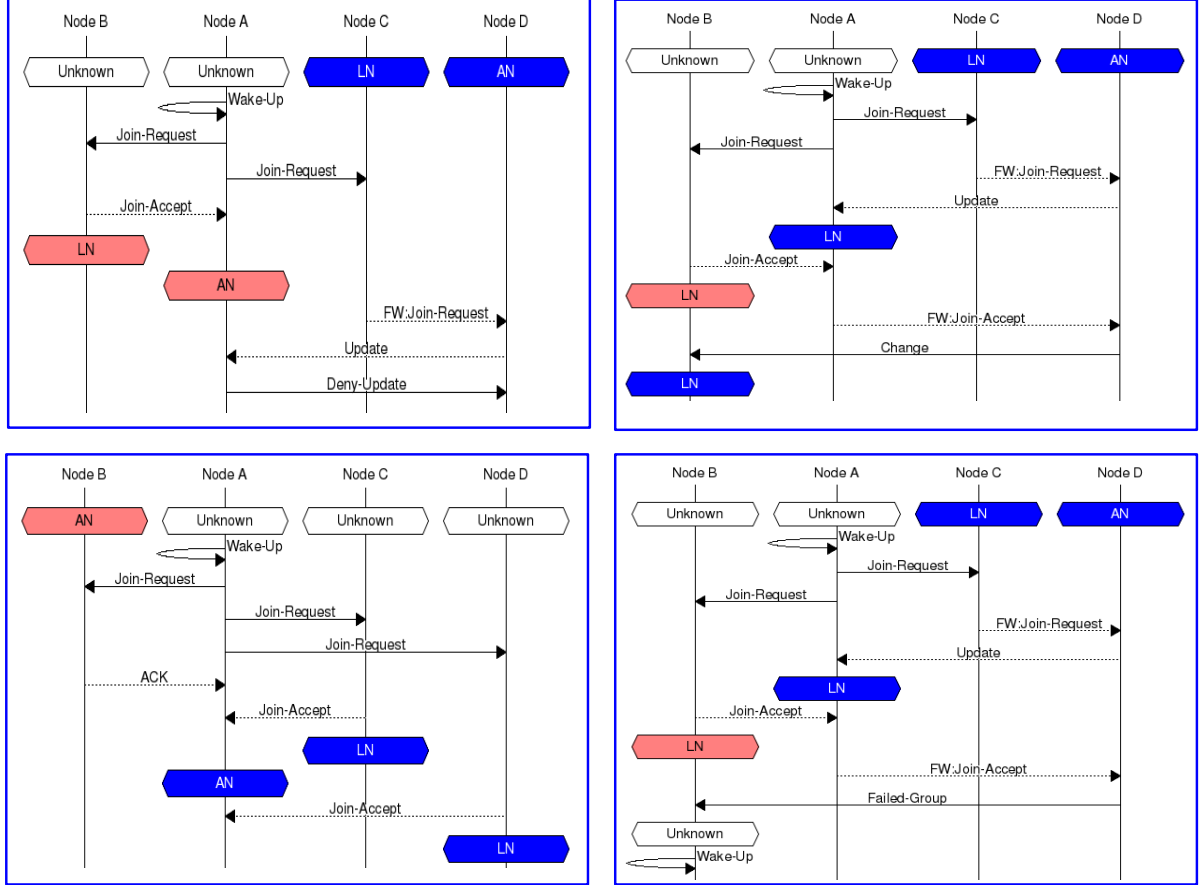


Figure 4.7: Some examples of interaction between distributed peers.

Step 4 (establishing the second level of the communicating groups): Similar to the second step, for each aggregate-node a **WakeUp** event is automatically triggered at random intervals bounded within a specific time frame. Upon occurrence of the **WakeUp** event each aggregate-node sends a message of type **Super-Node-Election** and its cached list of possible aggregate nodes to its neighbors. The neighbor nodes, which are the LN members of this aggregate-node, will also relay this request to their next level neighbor, while considering to avoid replication of forwarded information. When the receivers of the **Super-Node-Election** are the LN members belonging to the groups which are differentiated from the requester group, the election requests will be forwarded to the corresponded aggregate-nodes in those groups (see Figure 4.5). Figure 4.7 demonstrates some examples of interaction among distributed peers during group making and overlay construction.

4.3.2 Evaluation of the Group Creation Algorithm

The most important contribution of this work is to provide a load balanced clustering approach which efficiently works in a purely unstructured distributed environment to implement service based distributed system such as resource discovery protocols. This section aims to evaluate the algorithm's performance with respect to load balance, efficiency and scalability. We use OMNET++ and OverSim simulators to evaluate our proposal.

Table 4.2: Input parameter values used in the experiments (Scenario 1).

Parameter	Values
Physical network size	12, 24, 60, 120, 240, 600, 1000 nodes
Maximum size of a group in the first layer	40
Minimum size of a group in the first layer	10
Maximum size of a group in the second layer	20
Minimum size of a group in the second layer	2
Locality Factor k	$k=.025$
Number of iterations	100
Neighboring indicator N_i	1

The hierarchical overlay could be highly applicable to the deployment of such services if it can dynamically be adapted to the system requirements in a way that the overlay characteristics (such as the number of groups, replicas, super-nodes or aggregate nodes in each layer of the hierarchy) can be adjusted to suit the optimum number of values. For such overlay design, the support for load balance is an important and desirable aspect.

OverSim is a well-known validated P2P overlay network simulation framework which is based on the OMNET++ simulator. In order to validate our simulation results, we have used different dynamic random topology graph for each iteration with discrete uniformly-distributed random edge weights (e.g., in term of latency [1,100]). We have also tested, validated and verified the simulation results for various small-sized systems such as $n=12$, $n=24$ and $n=60$, where n represents the number of nodes.

To evaluate our work, in the first scenario we focus on the load balance issue. According to the simulation parameters in the Table 4.2 we conduct a simulation for a variable network size. We run the simulation in multiple iterations (runs) to better understand the overall behavior of the algorithm through large variations in the network topology for each iteration. For generality purposes, in each iteration, for the given network size (specified with the parameter n as the number of nodes in the network), we simulate a random network topology through generating a random connected graph with n nodes and l links (edges), where $l = \text{int}(1.6 * n)$ (l is an integer).

The clustering is performed along the way to satisfy the required conditions. As the input parameters we bind the group size in each layer in range while we maintain the locality factor constant. The proposed algorithm is supposed to generate a virtual hierarchical overly for a given physical topology of nodes (organized in a flat structure). In the first layer clustering, the nodes within the input topology must be divided into multiple groups (AN-Groups). Accordingly, in the second layer clustering, each group of AN-Groups create a SN-Group by electing a super-node among themselves. For each simulation run, the generated virtual hierarchical overly can be specified in output by determining values of 3 variables for each node including module-role, QMS-ID and SQMS-ID (each node can be specified by a constant integer, node-ID). Examples of this have been presented in Figure 4.2, where for node N3 (node-ID=N3) these variables are specified as module-role=LN, QMS-ID=N5 and SQMS-ID=N11

and for node N11 (node-ID=N11), module-role=SN, QMS-ID=N11 and SQMS-ID=N11. It also can be seen that one SN-Group, containing two AN-Groups have been created in output. In our simulation, we initially set all these variables for all nodes as unknown. The simulation proceeds through the (distributed) algorithm running at each node, until all variables for all nodes are adequately determined with respect to the grouping conditions (given as input parameters for simulation) in terms of locality factor and group sizes in both layers (AN and SN layers). The group sizes are optimally required to be within the desired ranges (given as simulation parameter). The members of each group are also required to fulfill the locality condition (as another input parameter for simulation). As we previously mentioned in Section 4.3.1, the locality factor is a parameter to specify the distance (in terms of latency, number of hops or hop counts) between each two members of a group (either AN-Group or SN-Group). In our simulation we define locality factor as a function of hop counts, where $k = \frac{1}{MHC}$ and MHC is the maximum number of hops between each two members of the group. But this doesn't limit the generality of our evaluation, since we can describe the locality factor as a function of (maximum) latency between each two members of a group and also the overall operation of the algorithm for both descriptions of the locality (based on hop counts or latency) are identical.

We measure the average load per node which is defined as the average number of transacted messages (number of messages sent or generated) by each node during the clustering procedure in different simulation runs. We also ignore to count the self-initiated messages, since these messages are not involved in real-world communication and interactions among nodes.

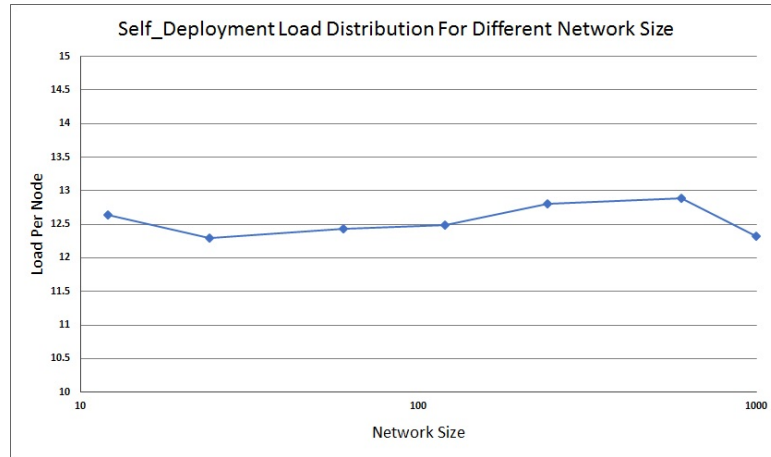


Figure 4.8: Evaluation results for the Load distribution in different network sizes (average number of processed messages per node for overlay making).

The simulation results are illustrated in Figure 4.9, demonstrating that the distribution of the communication load for the overlay setup are almost balanced between all the individual nodes in the system of size 600 and 1000 nodes and there are few nodes which receive and process the higher amount of messages. The clustering mechanism does not suffer from bottleneck and a single point of failure since the method does not rely on the specific pre-configured nodes. As we see in Figure 4.8, the average number of transacted messages by each node during the clustering procedure is between 12 and 13 transactions for the different network sizes. The processing nodes in the system can simultaneously and in parallel process

their received messages. In other words, it means that the load distribution is independent from the network size, thus, the system provides good scalability, while we increase the number of involved nodes.

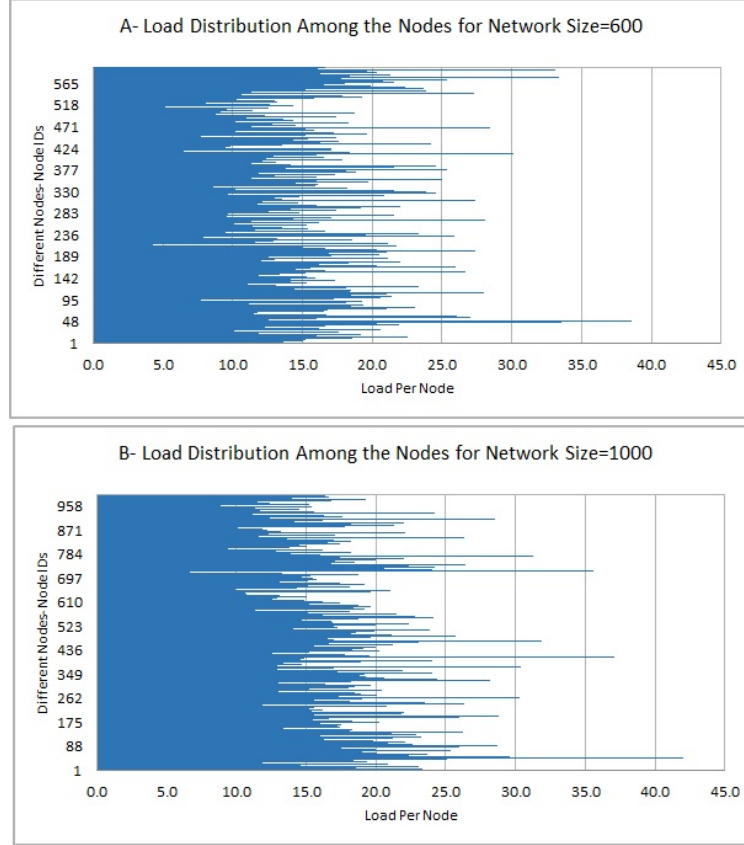


Figure 4.9: Evaluation results for the Load distribution among the nodes for network size=600 nodes and network size=1000 nodes.

In the second scenario (see Table 4.3), we evaluate the efficiency of the algorithm while considering the size of groups created and the locality of the nodes within each group.

Table 4.3: Input parameter values used in the experiments (Scenario 2).

Parameter	Values
Physical network size	12, 24, 60, 120, 240, 600, 1000 nodes
Maximum size of a group in the first layer	20
Minimum size of a group in the first layer	10
Maximum size of a group in the second layer	10
Minimum size of a group in the second layer	5
Locality Factor k	$k=0.10, 0.20, 0.33, 0.50$
Number of iterations	100
Neighboring indicator N_i	1

As we mentioned earlier, we have defined the locality k as an indicator parameter that shows how much the nodes within a group are close to each other (in in terms of hopcounts).

It is assumed that all the nodes within the logical groups created through our proposed overlay clustering mechanism are qualified members in terms of locality conditions, since the locality factor for each new group's member in every layer will be checked upon arrival of the request in the corresponded aggregate-node or super-node of the target group. So, It is guaranteed that all the nodes, within the groups created, are locality-aware, due to the required proximity conditions of the overlay. However, the size of all groups created might not satisfy the overlay design requirements. We define the Grouping Efficiency criteria according to the following formula:

$$\text{Grouping Efficiency} = \frac{\text{Number of the Qualified Groups}}{\text{Total Number of the Created Groups}}$$

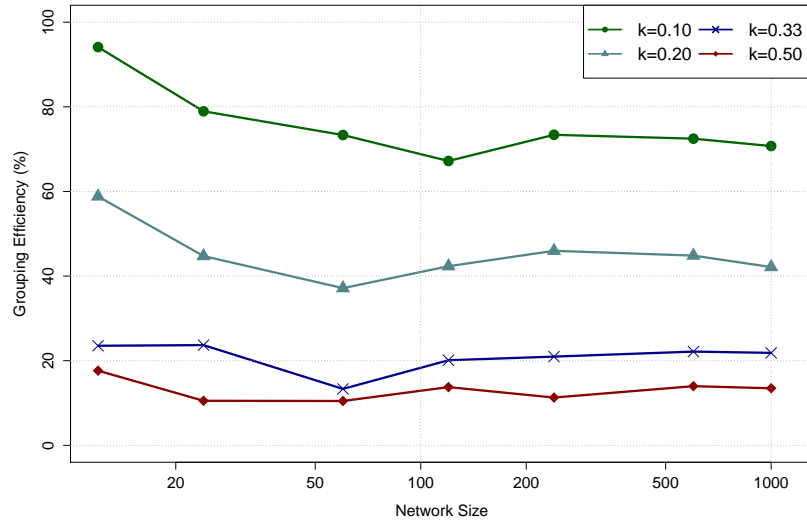


Figure 4.10: Evaluation results for the clustering efficiency for different locality factors and network sizes.

In this scenario we keep constant the values for the maximum and minimum allowed size of each group while we change the value of the locality-factor as well as the network size. The results in Figure 4.10 show that the algorithm provides better efficiency for lower values of the locality factor. As it can be observed in the figure, for each given network size, if we increase the locality factor the grouping efficiency will decrease. This happens because, for the larger amount of locality factor, the group diameter (maximum hopcounts) will be smaller (locality factor and group diameter have an inverse relation) and this will reduce the freedom degree of the algorithm to efficiently create qualified groups with respect to the given clustering conditions (in terms of group size and locality). The freedom degree for creating a group means the number of options (nodes) that can potentially be added to a group as new members while preserving a maximum allowable distance among members.

However, the algorithm is still efficient for the proper values (moderate values) of the locality factor. Setting the locality factor for very small values would possibly result in inefficiency for launching the services, launched on the top of the overlay because of the

expensive communication cost among the service components in the very large communicating groups. Figure 4.10 also demonstrates the scalability of the system where the increment of the network size doesn't almost have a considerable impact on the grouping efficiency.

4.4 Resource Discovery in Many-Core Systems

In this section we explain the general operations of our proposed resource discovery approach particularly for many-core-enabled cloud-like systems. We further discuss details of the proposed mechanisms and algorithms later in this chapter.

Our protocol makes use of a hierarchical structure, albeit relying on Chord based [406] [104] DHTs, to maintain resource information. By resources, we consider all the entities that compose a computational system, such as its Random Access Memory (RAM), CPU, Network Interface Controllers (NICs), interconnects (or network links) and graphic cards, as well as their operational status and performance metrics. We aim to make this information potentially available to all systems in a heterogeneous, many-core environment so that the processes are distributed in the most effective and efficient manner. Also, the decision can be taken at each core, in a truly distributed fashion.

We consider DHT as an essential technology that allows the design of robust distributed components providing storage and lookup services. Despite the existing master/slave data replication architectures often used today, DHTs are more reliable and able to provide performance guarantees [407] in fully distributed systems. In multi-core environments we attempt to avoid the exchange of unnecessary resource discovery information between nodes, thus keeping overhead to a minimum. To achieve this, we concluded that basic DHT approaches are not enough, and we decided to organize resources in multiple layers, implemented by using hierarchical Chord based [406] DHTs, essentially using resource aggregation and summarization.

We store the resource information (discussed in Chapter 3) in different layers using ring-based distributed hash tables, where resources are placed in a ring according to their hashed keys. With these hashing functions we can map all attribute values in a specific layer to the same m -bit space in a DHT ring. It must also be taken into account that depending on the resource discovery approaches, it is not necessary to use DHTs in all layers. Specifically for the upper layers we can store resource information in the format of the layer stamps, which can be validated according to a set of conditions in the incoming query templates. In our system we use different procedures at different layers by efficiency/complicity trade-offs.

Due to the system architecture definition, discussed in section 4.2.3, there are 3 different types of nodes: super-nodes, aggregate-nodes and leaf-nodes. For the discussion in this section, we assume that in many-core system, each physical core is represented by a node. (i.e., a node is supported by a physical core.) Accordingly, the positioning of these node types in layers within the hierarchy is the following:

Leaf-nodes are in Layer _{l_n} , so all the nodes in this layer maintain their own resource information plus a finger table, which handles forwarding Lookup request to other nodes in its DHT ring. Leaf-nodes have the ability to run the resource discovery procedure, which generates and sends a resource request query that includes an m bits key to an aggregate-node that hosts a QMS, a service which is initially described in Section 4.2.3 (We discuss this further in the remainder of this section). Leaf-nodes communicate with their aggregate-nodes and the aggregate-nodes establish a structured DHT-based overlay in the form of a Chord ring among their leaf-nodes. For instance, all the cores in a CPU can be considered as leaf-nodes,

bearing in mind that there is at least one core per CPU, which runs a service that behaves as the aggregate-node, a central contact point that summarizes all the information in that CPU.

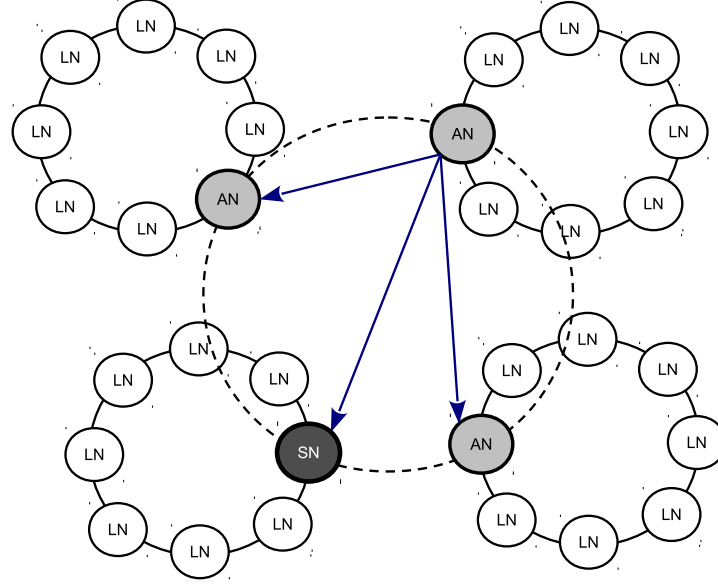


Figure 4.11: Types of nodes in our hierarchical structure.

Aggregate-nodes address the information on Layer_{an} . QMS instances are the information service modules that reside in aggregate-nodes. QMS gets a query from a leaf-node and then starts the lookup procedure to find the required reply. If the reply is found in the local QMS domain, it will be sent back to the requester. Otherwise the QMS forwards the query to other aggregate-nodes in its neighborhood. It will use a probability based method to select the next aggregate-nodes among others. In fact, each aggregate-node in our many-core system is a QMS host and in this chapter we may use “QMS” to refer to the aggregate-node.

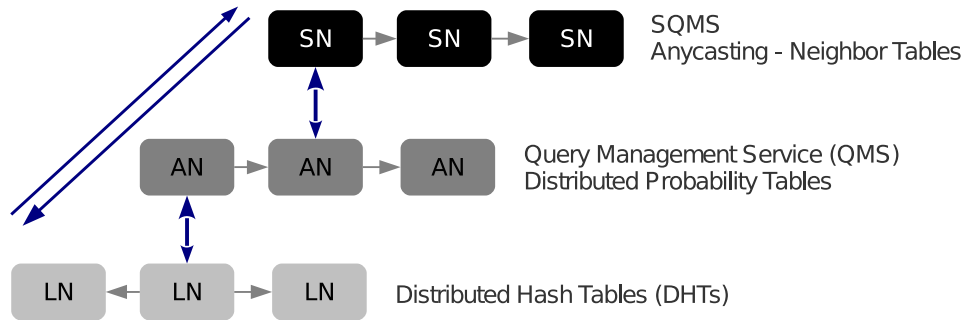


Figure 4.12: Transaction of the query messages between layers.

Super-nodes are in Layer_{sn} . If the resource discovery cannot find a specified resource after searching all the QMS in the local board, the query message will be forwarded to other neighbor nodes in the network by an SQMS which is hosted on the super-node. This super-node has

information about all the adjacent super-nodes in the network. Accordingly, it uses Anycast mechanism to select the subsequent super-node among others, aiming for minimum latency (see Figures 4.11 and 4.12).

According to the characteristics of each resource, we can extract the values for our set of attributes. Conforming to the resource requester's priority list of the attributes, multiple attributes values of each processor are extracted and encoded in order to form an m -bit key which is used to update and locate that resource. In the system with $d1$, $d2$ and $d3$ attribute dimensions, in each layer, we need to generate a set of keys ($k1$, $k2$ and $k3$) with $m1$, $m2$ and $m3$ bits for all the layers. A query message consists of three parts (c_1 , c_2 , c_3) or (c_{ln} , c_{an} , c_{sn}), each representing a set of conditions for satisfactory resources in each layer. The below script depicts the general format of the resource discovery query messages.

Query : $Condition1 \wedge Condition2 \wedge Condition3 \wedge \dots$
Condition : $AttributeOperatorValue \wedge AttributeOperatorValue \wedge \dots$
Operator : ($<, \leq, =, \geq, >$)

We ensure that any leaf-node is able to discover other resources by using the resource m -bit keys (for different layers) up to a certain limit of time with the order of $\log N$, where N is the number of leaf-nodes in $Layer_{ln}$. In the inter-core layer, discovery proceeds in a multi-hop fashion with each of the leaf-nodes maintaining its small finger table containing the information about other leaf-nodes in the group. They forward the discovery message recursively to the leaf-node that is closest to the key of the original requester's query.

Algorithm 4 describes the general process of resource discovery in a large cluster with multi-dye and multi-core nodes. We discuss further different steps of this algorithm later in this chapter. The QMS components, maintain the description values for all the keys for which it is responsible, and for each core there is only one unique key. QMS specifies how keys are mapped to cores, and how a node can find data for a specific key or range of keys by first locating the nodes which are responsible for that particular key. The summarization of the QMS responsibilities and functionality is as follows:

- Start-up, building DHT overlays and gathering information.
- Entry point for the ring lookup (Requesters send their resource queries to their local QMS).
- Maintaining resource cost table (i.e., probability tables), retaining information about other QMSs which reside on the other aggregate-nodes in neighborhood such as neighbor ID, probe factor and the Inter-Chip layer information. Probe factor is the probability of finding a specific requested resource in a remote QMS domain.
- Retaining the local Inter-Chip layer information, a summarized set of characteristics shared among all the leaf-nodes in a group.
- Sustaining information about the local super-node which is the representative of several leaf-nodes groups in the system.
- Processing, managing and forwarding of the query message between layers.

At $Layer_{sn}$, the super-nodes are able to send the query to a group of super-nodes in vicinity that are using the same anycast address. The actual number of receivers can range from only

Algorithm 4: Resource discovery general procedure.

```
Input: applicationArguments                                /* description of the required resources */

Output: discoveredResources                               /* list of discovered resources */
Data: RCT                                                  /* resource cost table */

Data: reqMsg, repMsg, QMS, remoteQMS                       /* query management service */

Event: discoveryEvent; discoveryEvent.setTrigger;
Collection: manyCoreSystem:=Set{n, f(i)}                  /* n nodes, f(i) cores per nodei */

Collection: multiCastGroup, qRes(qualified recuses), dRes(discovered resources);
foreach node ∈ manyCoreSystem do
    node.nodeAssign(QMS);
    foreach core ∈ node.requesters do
        core.coreAssign(discoveryEvent, RCT);

On-discoveryEvent: querymain=generateQuery(applicationArguments);
core.set(original-requester);
QMS=getLocalQMS(coreOR);
reqMsg=generateReqMsg(querymain,coreOR);
send(reqMsg,QMS)                                          /* send query from the source core (original requester) */

On-receive(QMS):
subQueries=queryAnalyzer.divide(reqMsg);
foreach query ∈ subQueries do
    forward query to the proper layer if it is required;

if QMS.layerproperties satisfies the query then
    results=QMS.Lookup(query);
    if query is fully resolved or query is expired then
        send(results,QMSOR);

    else if query is partially resolved then
        send(results,QMSOR);
        forward query to the proper layer if needed;
        remoteQMS=RCT.selectNextQMS();
        send(updated query,remoteQMS);

    else
        forward query to the proper layer if needed;
        remoteQMS=RCT.selectNextQMS();
        send(query,remoteQMS);

else
    forward query to the proper layer if needed;
    remoteQMS=RCT.selectNextQMS();
    send(query,remoteQMS);

On-receive(SQMS):
forward query to the proper layer if needed;
multiCastGroup=SQMS.getNeighborGroup();
Broadcast(query,multiCastGroup, Anycast);
On-receive-queries-results(QMSOR):
qRes.add(querymain, results);
if discovery is completed or main query is expired then
    dRes=queryAnalyzer.converge(querymain, qRes);
    repMsg=generateRepMsg(querymain, dRes);
    send(repMsg,coreOR);
```

one to several group members, where some of the receiver nodes satisfy the query requirements for metrics such as latency, availability and load balancing between the target resources.

4.5 HARD Mechanisms

4.5.1 Initialization Phase

Initialization procedure is the pre-processing phase of the resource discovery process, where the initialization of the environment variables (e.g., modules configurations) is performed. In fact it is necessary in order to provide the minimum requirements for the execution of the discovery algorithm. The discovery process in the next phase is performed on the basis of these primary settings which consist of the resource grouping and clustering indexes, module roles initialization and allocation of the primary values for the underlying data structures. Some of these settings are directly calculated and stored in the local memory of each node during initialization phase while some others are dynamically updated/recalculated according to different policies and during the discovery procedure. There are also system variables which are required to be set by the system administrator such as grouping policies and grouping thresholds.

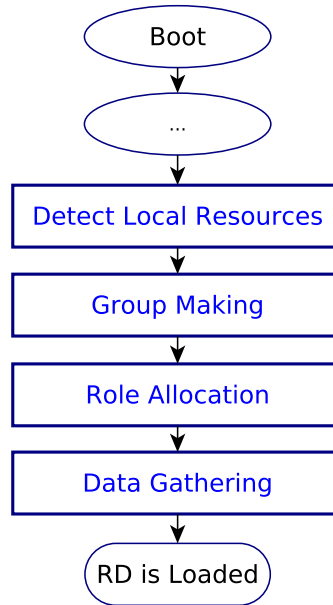


Figure 4.13: Initialization phase.

In summary, the initialization phase (see Figure 4.13) of the resource discovery modules consists of the following steps:

1. Self-organization and self-stabilization of the multi-layers communication overlays (zero/auto-configure overlays)
2. Distributed role-allocation (e.g., leaf-node, aggregate-node, super-node)
3. Data gathering and registration (initialization of the successor, probability and neighbor tables)

The details of the algorithm for self-organization and self-configuration of the nodes in hierarchical layers (e.g., $layer_{ln}$, $layer_{an}$ and $layer_{sn}$) was presented in Section 4.3.

4.5.2 Data Structures

In HARD3 initialization phase, according to a distributed self-configuration mechanism, each single HARD3 instance (running on different vnodes) obtains its own instance role (i.e., module-role), which clarifies the future operational behavior of that module instance in terms of discovery. Depending on the module role, each resource discovery instance is responsible for maintaining and updating a set of information in memory, which are the following:

- The nodes which have the LN role maintain a successor table, a resource state table (i.e., a vector of states for all the resources belonged to a vnode), a leaf-stamp, a pointer to the sibling node (i.e., a vnode with similar type of resources in the current DHT-ring) and a QMS-ID. Successor tables in leaf-nodes are created through getting information from the system resource description provider and by leveraging some dynamic algorithms. The leaf-stamp is a key that demonstrates all the characteristics of a computing node (e.g., a processor) in the leaf-node layer and it is generated by extraction and aggregation of the predefined layer's characteristics presented by the system resource description provider. The QMS-ID also specifies the address of a cluster representative which the current leaf-node belongs to (i.e., the address of an aggregate-node in the system which provide RP-QMS service).
- The nodes which have AN role (i.e., the nodes with QMS functionality) maintain all the information related to the LN layer as well as a probability table, an aggregate-stamp and a SQMS-ID (i.e., Super-Node ID). The probability table will be created and configured by the aggregate-node itself during initialization phase and it would be updated during the resource discovery procedure due to the dynamic behaviors of the HARD3 module instances which are running on the other aggregate-nodes in the system. Furthermore, the aggregate-stamp indicates all the characteristics of a computing node in the AN layer through extraction and aggregation of those properties from the resource description provider.
- The nodes which have SN role (i.e., the nodes with SQMS functionality) maintain all the information related to both of the LN and AN layers as well as the neighbors table and the node-stamp. The neighbors table provides information about the other super-nodes in vicinity and the node-stamp indicates all the predefined characteristics of a computing node in the SN layer.

It needs to be taken into account that all the instances of the resource discovery modules must have initial states either by using static configuration or performing the initialization procedure (resulting from a dynamic self-organization of the logical network overlays in the hierarchy). The value for module-role can be LN, AN or SN. We must also note that, each one of the module instances has the capability to act as a leaf-node by default. However they can not play the role of aggregate-node or super-node unless their module-roles clearly are marked as such. Moreover, the role of each module instance can be changed dynamically during the resource discovery procedures, and system self-organization.

4.5.3 Resource Requester and Resource Information Provider

RRs are the module instances that query Resource Information Providers (RPs) on behalf of HARD3 users (i.e., the resource discovery users or the system components such as resource

manager or process manager which need to discover resources for purposes like resource allocation) for their needed resource information and the RPs are the module instances that provide information services to other RPs and RRs. As it is shown in Figure 4.14, RR operations can be summarized according to the following steps:

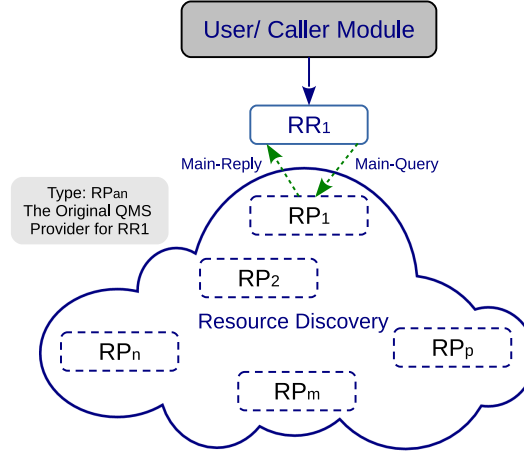


Figure 4.14: Resource requester general behavior.

(1) RR reads memory to get the description of resources which are demanded by a user (reading and analyzing the user's query description). In accordance with our hierarchical resource description, the description of the required resources can be accurately reflected in a flexible query description. A general query might be described as a single group of heterogeneous resources, which contain several homogeneous group of resources. The description of each homogeneous group represents the group characteristics such as number of desired resources, static and dynamic desired properties of resources in each standard layer and the required inter-resources and inter-groups communication properties.

(2) Using the query description set by the user, RR creates a main-query message and sends it to its local QMS. The local QMS is the default self-configured RP for RR. Each local QMS, on behalf of its RR clients, manages all the relevant discovery steps for a main-query in the distributed system.

(3) Later, RR receives a main-reply message corresponding to its discovery request. This consists of information on the discovered resources which are pre-reserved for the user (i.e., application segments belonging to the user). RR is able to either release them or reserve these resources for a longer time period. The discovery temporary reservation for each resource will automatically end after a certain time period if the related RR makes no decision on the reservation policy. Moreover, a RR will release the reserved resources when those resources are not needed any more by the user (i.e., application execution is terminated).

Unlike the irrelevancy of RR behavior to the module's role, the RP algorithms and mechanisms are mostly dependent on the module's role. For this reason, we elaborate on the details of these algorithms in the next sections, explaining the behavior of the HARD3 modules from two different viewpoints: RRs and RPs.

4.5.4 Dynamic Aspects

Dynamicity poses some extra challenges to the general resource discovery problem since it requires keeping track of those changing resources. For this reason, in large scale distributed many-core environments, where all the nodes continuously change their behaviors and properties, resource discovery will be a critical component, as an inefficient discovery method would unnecessarily consumes a significant portion of network bandwidth as well as computing resources. Providing a dynamic solution for resource discovery involves the following aspects:

- Dynamic changes of collective or individual attributes: In our resource discovery solution we considered that resources have static and dynamic attributes. We store static features of resources in distributed hash tables, probability tables, neighbor tables and layer stamps, taking into account that this information is not necessary to be frequently updated. On the other hand, we calculate and temporarily store the dynamic properties such as communication characteristics inside a group of required resources, on demand and during the run time discovery procedure in certain nodes that are providing distributed information service. However, in the system, other than dynamic individual or collective properties of resource/resources, there are some other aspects of dynamicity, which have to be considered in our work.
- Removing an existing instance or adding a new instance: Our resource discovery system has been built based on communication and information exchange between a large numbers of peers, where each peer has its own resource discovery module instance. During system runtime, it might happen that some of the peers fail and after a while they become recovered. When an instance fails, the first peer which sends a request to the failed peer will detect the failure and will inform the local QMS in the cell. Depending on the instance role of the failed peer the system recovery procedure will perform one of the following actions:
 - If the failed peer is LN, then the local QMS of that peer, reorganizes the related successor tables in the resources, which they have pointed (referred) to the failed peer by removing its corresponding pointers and replacing it with other successor nodes.
 - If the failed peer is AN or SN, then the requester which determines the failure will send an announcement to its alternative QMS/SQMS and informs it about the failure. The corresponding QMS/SQMS appoints another resource information provider in the local cell as an alternative QMS/SQMS and transfers the necessary information such as probability table, neighbor table, etc., to the appointed node and finally it announces the new QMS/SQMS to its members in the local cell.

When a new instance joins the system, it will first recognize its grouping cell by asking its neighbors and then it will get the information about the local QMS/SQMS. In the next step, the new instance will send a registration request to its QMS where this request will be answered with a primary role for the new joined peer. Due to the assigned role, the new instance will collect the necessary information. And finally QMS informs other peers in the cell about the existence of the new member.

- Dynamic changes of module instances roles or underlying communication overlays: The primary role of each one of the resource discovery instances in the system is not fixed and

due to the different conditions and situation the peers in the system may change their role, communication overlay or layer. The primary peer role or overlay comes from the result of a distributed election mechanism inside the local cell during resource discovery booting or initialization phase. But after that, and during system runtime, the local QMS will decide about changing the role or overlay for every one of members in the local cell. It means that QMS will assign new role or overlay for the members in certain situation such as joining, leaving or failing of the nodes and there is no need to use election mechanism.

- **Hardware changes:** In case of dynamic hardware changes (joining new hardware-leave or failure of an existed hardware), resource discovery provides the updated information of the current hardware to its clients such as process manager. It means that the result of resource discovery has to be accurate considering all the possible changes in the underlying hardware. Therefore resource discovery provides mechanisms to detect the failed resources and it does not put the expired, crashed or departed hardware resources in the final list of the discovered resources. Any changes in the hardware architecture (such as memory and processor) or in the communication hardware infrastructure (such as interconnected networks and network adapter) will affect the information contents which are spanned among the resources around the network and it assists the resource information providers to provide accurate information service.

4.5.5 Communication Events

After the initialization phase has been completed, system actions are concentrated along the line of maintaining resource information, handling queries and managing unpredictable changes in the system configuration. In order to handle queries and route discovery requests to the proper resources within and across layers, we have defined several types of events (see Figure 4.15 and Table 4.4) where each event specifies the type of communication and also the necessary actions that are required to be done by a receiver node. The receiver node basically acts as an event handler responsible for handling the triggered event. These events are the following:

- **Lookup Event:** DHTs have been used to store the highest details of the resource information in the lowest layer of the resource description hierarchy, therefore all the peers in a mini-cell (i.e., a group of vnodes with common QMS-ID and SQMS-ID) participate in a ring-based DHT. It is not necessary to have a single or flat DHT, rather in each mini-cell, for each dimension, a flat/hierarchical ring can be used. RPs trigger **Lookup** events in order to search the entire leaf-nodes in the local mini-cells for the desired resources.
- **Update Event:** Once a query is successfully resolved in a vnode, an **Update** event is triggered in order to inform the original resource requester and all the intermediate resource providers on the result of the corresponding query. The response (**Update**) message follows the reverse path taken by the query message through the overlay network and updates the values for the resource-type-depended variables such as probe factors and preferred-nodes in the probability tables for the next usage. Generally, the leaf-nodes return updates with the whole results, with partial results, or with no results.
- **Upward Event** is an inter-layer communication event received by a SQMS provider in the super-node layer. It notifies the SQMS that the lower layers in the current cell

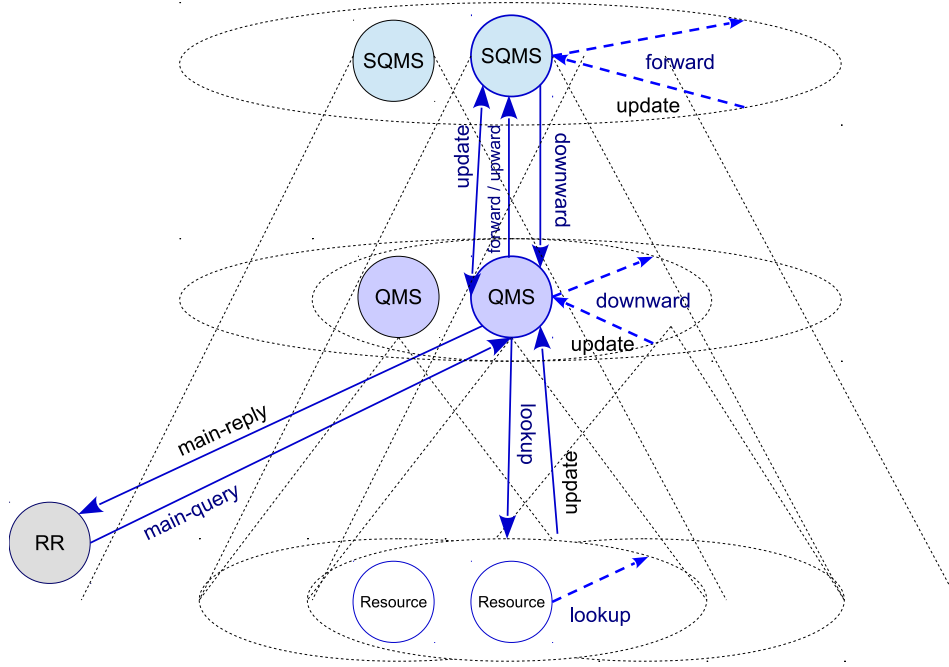


Figure 4.15: Communication events within layer and across layers.

were already explored with no result, and the discovery process must be continued by directing the query to other SQMS in the system.

- **Downward Event:** In order to resolve a query, due to the hierarchical structure of the resource information and query descriptions, each query starts the discovery process from the higher layer and when the resource description in a layer satisfies the query conditions for that layer, the query must be sent downwards to the lower layer which supports adequate detail information. In fact queries are traversing between layers to extract more accurate information of the desired resources. In other words, a **Downward** event is generated by a RP in an upper layer and it will be send to another RP in the lower layer, so the recipient RP can be sure that the query conditions in the higher layers already have been achieved.
- **Forward Event** is the major communication event in the super-node layer, which alerts the receiver that the lower layers of the current cell are not explored yet. It is obvious that the lower layer exploration is only required if the query conditions in the super-node layer are met. **Forward Event** can also be generated by the original QMS of a query in the lower layer and received by the SQMS in the upper layer.
- **Main-Query Event:** Once a HARD3 instance, running on a particular processing node, receives a discovery demand from a user, the RR component starts by generating a main-query. This happens when the local processing resources are not sufficient for an efficient application execution, and some extra remote resources are required to improve the execution performance. The resource requester will trigger the Main-Query event in a target node in its local mini-cell which provides query management service (i.e., QMS).

Table 4.4: Inter-layer and intra-layer communication events.

Event	Source Event Generator			Target Event Handler			Description
	LN	AN	SN	LN	AN	SN	
Lookup	yes	yes	yes	yes	yes	yes	DHT lookup in layer _{ln} , Intra-layer communications in layer _{ln} , layer _{an} – > layer _{ln} (inter-layers communications)
Upward	no	yes	no	no	no	yes	Ensuring that the lower layers in the current cell (a group of <i>vnodes</i> with common SQMS-IDs) have already been searched with no results, layer _{an} – > layer _{sn} (inter-layers communications)
Forward	no	yes	yes	no	no	yes	There is no assurance that the lower layers of the current cell are explored, Intra-layer communications in layer _{sn} , layer _{an} – > layer _{ln} (inter-layers communications)
Downward	no	yes	yes	no	yes	no	Ensuring that the query conditions are met in the upper layer (layer _{sn} but the query is still not fully resolved and further search is required to be carried out in the lower layers., Intra-layer communications in layer _{an} , layer _{sn} – > layer _{an} (inter-layers communications)
Downward_{cbk}	no	yes	no	no	yes	no	Ensuring that all the potential children of the current vertex (i.e., an aggregate-node or a <i>vnode</i> with QMS functionality) in the layer ₂ search tree have already been explored. Further search is required by calling back to the parent of the current vertex. The probability table in receiver nodes must be updated. Intra-layer communications in layer _{an}
Update_{qms}	yes	no	no	no	yes	no	The result (i.e., full result, partial result or no result) of a Lookup query is delivered to the caller entity which is the QMS of the event generator (event initializer), layer _{ln} – > layer _{an} (inter-layers communications)
Update_{sys}	no	yes	no	no	yes	yes	Ensuring that a query (sub-query) is completed (with partial or full results). The result is delivered to the original QMS requester (i.e., the QMS which has registered the original query). The intermediate entities must be updated. Intra-layer communications in layer _{an} and layer _{sn} , layer _{an} < – > layer _{sn} (inter-layers communications)
Update_{vic}	no	yes	no	no	yes	no	The QMS of the aggregate-nodes in the vicinity of the event generator are required to be updated on their knowledge of source of resource. Intra-layer communications in layer _{an}
Update_{nul}	no	yes	no	no	yes	yes	Ensuring that a query (sub-query) is unusually completed (with partial or null results). This could happen when all the potential <i>vnodes</i> in the system are explored or the query is expired. The result is delivered to the original QMS requester. Updating is not required for the intermediate entities. Intra-layer communications in layer _{an} and layer _{sn} , layer _{an} < – > layer _{sn} (inter-layers communications)

- **Main-Reply Event:** Upon completing a main-query a main-reply will be created by the original QMS of the main-query to inform the HARD3 user on the overall result of the discovery request. Depending on the states of the sub-queries, the discovery reply for a main-query may contain full results, partial results, or no results.

4.6 HARD Algorithms

4.6.1 LN Algorithm Description

RP_{ln} s (i.e., vnodes with the module-role of LN) operate as following (see Algorithm 5):

(1) Upon receiving a **Lookup** request in a receiver node which is denoted as RP_{ln} , it checks its local resources in order to find an available match/matches that meets/meet the requester criterias described in the **Lookup** message. For doing this, the RP_{ln} verifies if its leaf-stamp is validated either in terms of equality or similarity (based on the degree of similarity returned by a similarity function) by the lookup-key (i.e., the description of the query conditions for the $layer_{ln}$) mentioned in the **Lookup** message. It must be taken into account that the acceptable similarity degree to resolve queries can be a constant value for all the queries, predefined in the entire system, or it can be variable for each query.

Algorithm 5: Processing a Lookup sub-query by a RP_{ln}

```

Input: sq= The received Lookup sub-query
/* sq, temp:sub-query, ft:finger-table of  $RP_{ln}$  */
/* for a sub-query, nRR is the number of resources requested, nDR is the number of already
   discovered resources, DRs is the list of IDs for already discovered resources */
temp=sq; temp.nRR= sq.nRR- sq.nDR; temp.nDR=0
idx=ft.get-entry-index(temp.cln) // checking the lower bound of the finger table ft, if idx==NULL =>
    there is no an entry index for temp.cln in the finger table, an entry index refers to a
    possible range of key values defined in ft, ft specifies a successor-node-id for each entry
    index
if  $RP_{ln}$  is qualified due to temp.cln then
    matched-resources=check-local-resources(temp)
    temp.DRs.push(matched-resources); reserve(matched-resources, temp)
    temp.nDR=matched-resources.size(); sibling-node=hasNextSibling()
    if (sibling-node)  $\wedge$  (temp.nDR < temp.nRR) then
        temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
        send(Lookup, sibling-node, temp)
    else
        temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
        temp.preferred-vnode=check-sor-capabilities(temp)
        temp.visited-qms-ids.push(qms-id)
        send(Updateqms, qms-id, temp)
else if ( $RP_{ln}$  is not qualified with respect to temp.cln)  $\wedge$  (idx) then
    temp.nDR=0; temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
    temp.visited-qms-ids.push(qms-id)
    send(Updateqms, qms-id, temp)
else
    if (vnode-id==maxRank)  $\vee$  ( maxKey < temp.qls) then
        /* checking the upper bound of the finger table ft */
        temp.nDR=0; temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
        temp.visited-qms-ids.push(qms-id)
        send(Updateqms, qms-id, temp)
    else
        next-node=ft.get-successor-id(idx)
        forward(Lookup, next-node, sq)
        /* sending the input sub-query to the next-node in the DHT without change */
return

```

(2) If all the resources required by the query are found in the local set of resources (i.e., if the query is fully resolved), the current lookup procedure exits and, consequently the RP_{ln} creates and sends an **Update_{qms}** message to the local QMS (i.e. the aggregate-node, providing the QMS service to RP_{ln} , that has initiated the current lookup procedure) containing the information on the matched resources in the current leaf-node. In other cases, if some partial

results are achieved or there are no resources available in the local set of resources, while the leaf-stamp of the current node is validated by the query's lookup-key, the RP_{ln} checks if it has sibling-node. On the existence of a sibling node, the **Lookup** message will be redirected to the sibling-node, while the content of message is updated for the partial results achieved in the current node and, if the sibling-node has not exist, similar to what is performed on a full query resolution, an **Update_{qms}** message containing partial result or no result will be sent to the local QMS. It must be taken into account that each lookup query contains information such as the number of resources required, the number of resources discovered and the properties of the resources desired in layer_{ln}. The receiver of a **Lookup** message identifies the number of resources, which are currently needed to be discovered due to the information extracted from the message content. This information includes the lookup history, which clarifies on the resources which are already discovered in other SoRs. Sibling-nodes are the vnode which provides similar resources. These vnodes are not directly participating in the DHT, rather among each group of sibling-nodes only a single member joins the DHT of the leaf-nodes in the current mini-cell. In fact sibling-nodes are the hidden members of the DHT, which are called whenever more SoRs of a particular type of resource are needed. Moreover, each group of sibling-nodes constructs a chain where each node knows only its single sibling-node. The first node of the chain directly gets position in the DHT while the last-node ends the search in the sibling-nodes.

(3) If the lookup-key fails to validate the leaf-stamp of the current node, thus, according to the DHT properties (i.e., the properties of the DHT ring which has been created through participation of a group of leaf-nodes in the LN layer) and the local successor-table of RP_{ln} , the lookup ending conditions would be examined to determine whether they are satisfied or not. Accordingly, there would be two possibilities: (a) If a lookup ending condition is reached, the lookup procedure ends, and the RP_{ln} sends an **Update_{qms}** message to its QMS address containing information, which ensures the local QMS about the non-existence of available matches for the lookup query constraints in the leaf-node layer, while the search space has efficiently been explored. (b) If no one of the lookup ending conditions is reached, the RP_{ln} redirects the **Lookup** message to the next RP_{ln} in the DHT by using its successor-table.

4.6.2 Probability-Based Discovery

In this section we begin to elaborate the RPs mechanism for ANs by describing an example for resource discovery in a multi-core-enabled cluster. In the next section we continue to discuss AN-RPs operations in a more generic manner. Figure 4.16 depicts the mechanism of resource discovery in a cluster with multi-core nodes. Finding the optimal resource for a query with minimal requirements depends on the processing capability, memory and availability. The search algorithm first explores possible neighboring nodes and selects the most promising ones, explores the search graph, and goes to the next tier only if the found optimal does not meet the minimal requirements of the query.

We have already classified the performance metrics and parameters to evaluate resources in the format of resource description in three individual layers: Core2Core, Die2Die and Node2Node communications level (see Table 3.3). These parameters handle gathering real time information about the current status of execution and processing capabilities of the available resources. Each group of processors has its own Resource Cost Table (RCT) (i.e., Probability Table), (see Figure 4.17), which includes metrics to assess other processors viability to be used as destination of process migration. According to the metric values in the RCT,

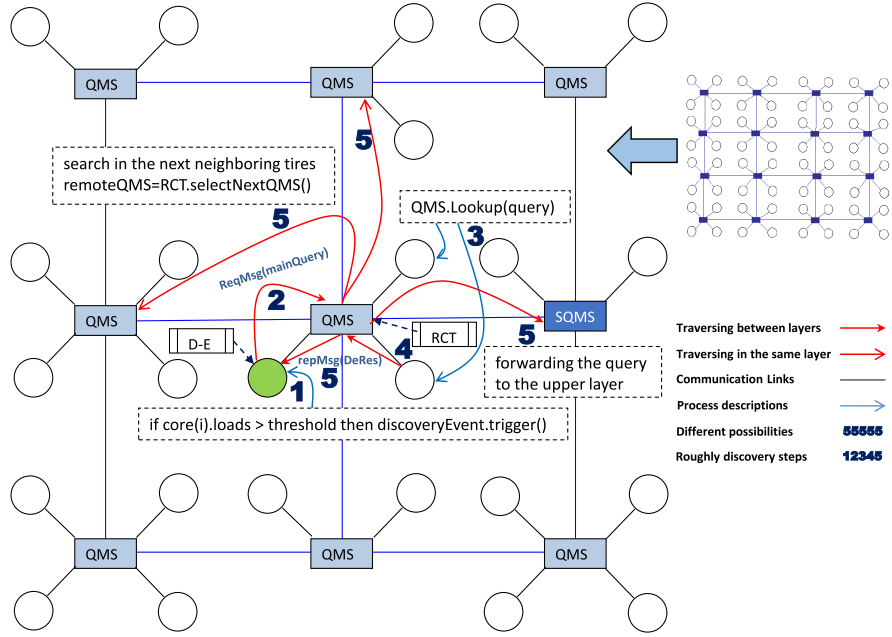


Figure 4.16: Mechanism of resource discovery according to the cost of resources (the numbers indicate the sequence of each message).

ranking algorithms generate a rank number for every particular processor in the network. Figure 4.17 is an example of RCT. According to a specific resource description, we must identify the relevant metrics for describing the potential performance of a given resource.

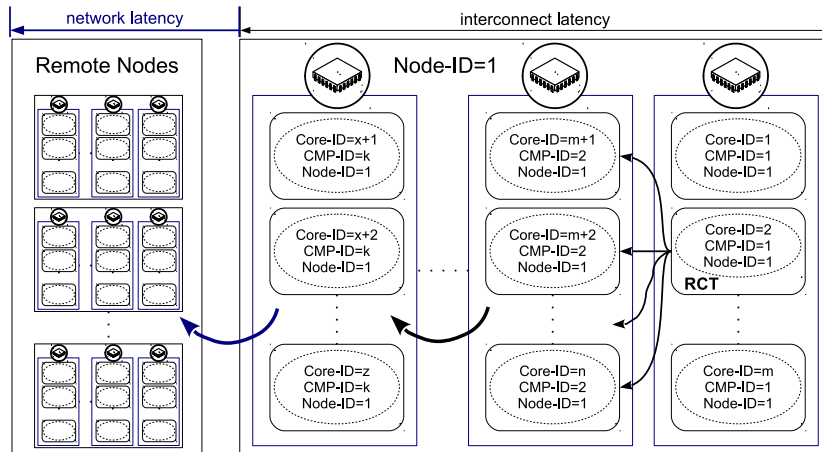


Figure 4.17: RCT- Assessment resources cost per type of query, for various ranges of latency.

Resource cost tables (i.e., DPTs) are maintained in aggregate-nodes and they keep information about the probability of finding a requested resource on different neighbor groups of nodes. They aim to evaluate the cost of resources per query type. In order to organize the RCTs we

need to define query types. RCTs include fields such as NID, K2, PF and Probability, where: NID is the neighbor ID; K2 is a binary key that represents the $Layer_{an}$ information; and PF is the probe factor for each neighbor. RCT ranks the neighbor QMSs for each queryTypeID which is conceptually elaborated and based on the range of computation and communication latencies.

Before performing query classification we must estimate the optimal or minimal application resource requirements. Computational clusters are common alternative platforms for handling massively large computational problems and parallel applications. Many-core systems are able to be more efficient when resource management is deployed according to the run-time application requirements. Generating queries for resource discovery requires a capability to predict the resource requirements of parallel applications before making decisions for scheduling. There are a multitude of technologies to estimate and extract the application features, which try to deploy performance models or mechanisms to forestall resource utilization in case of computation and communication complexities, for applications launched under a distributed parallel system, and including multiple homogeneous or heterogeneous resources with various processing capabilities [408, 409]. Application modeling or application description can provide accurate information for querying operations in the resource discovery protocol.

At the time of the query generation, depending on the application features, we can assign a queryTypeID for each particular query. After discovering an appropriate resource, all the query-resource mapping information is used to update the probability information and reuse of the discovered resources in similar query statements. The system have to build a different resource cost table for each query type separately. If we increase the number of resource types it helps the system to be faster and more accurate but it has the drawback of using more storage to keep instances of RCT per queryTypeID.

$$OnDiscoveryEvent : query = generateQuery(applicationArguments)$$

The cost of a resource depends on three aspects, the first one is the processor that is looking for other resources (i.e., the requester) to augment its processing capabilities and distribute jobs among them. It starts the discovery process and triggers its assigned discovery event. The second element is the application that runs on the aforementioned processor. There are a variety of applications with different specifications and requirements. The final element is the processor that would be nominated as the result of the discovery. Considering these elements and their effects on the resource evaluation, we can model the complexity of the assessment as follows:

$$ResourceCost = CommunicationTime + ComputationTime$$

The *ComputationTime* includes the execution time as well as the time for memory access.

In classic resource discovery protocols, a typical cluster environment is generally defined as a Virtual Organization (VO) which consists of two types of individuals: resources and resource directories. In our approach, for discovery we assume that a cluster (i.e., the QMS region) is a directed graph $Cg(Ve, Ed)$ with n nodes and w edges. Each edge $ed(a, b)$ connects to two nodes a and b where: $ed(a, b) \in Ed \wedge \forall a, b \in Ve \mid a, b \leq n$

We assign a probe factor Pf to each one of the links from local QMS to the neighboring QMSs. Nodes in the graph are cores in the real cluster. Cores are linked to each other through

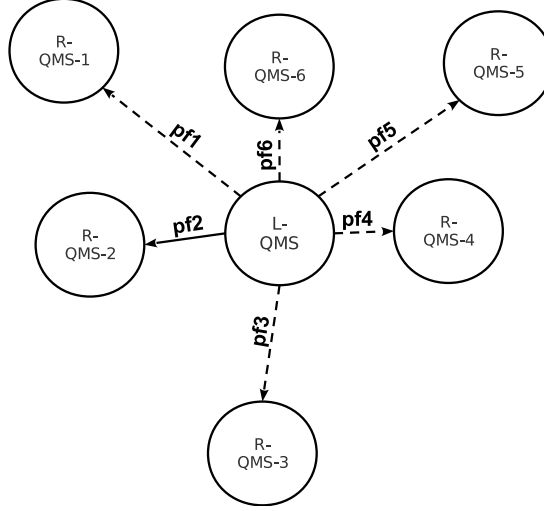


Figure 4.18: Selection of the consequence QMS based on values of probe factors and probabilities.

interconnection networks. We assume that all the cores which are placed in a CPU or cluster node are members of a QMS. The edge from a to b (i.e. $ed(a, b)$) is the connection link between core a and core b , where communication cost to describe it are bandwidth and delay. For each core, states and descriptions of its processing resource and also other group of cores in the first tier vicinity are stored in localQMS. The cost of resource r in the localQMS of core b through $ed(a, b)$ maintained as $Rc_{ab}^r = \frac{1}{Pf_{ab}^r}$, which Pf is the probe factor for the existence of resource r with resource type of t in the QMS of b when the query is going to be forwarded from the QMS of a (see Figure 4.18).

In our resource discovery approach, we generate one query per discovery request, and in the whole cluster system according to the number of requesters, several queries would be generated concurrently. They explore different paths in line with the link parameters like probe factor, and the values of the probe factors are updated by all those query replies that have completed a discovery procedure.

Queries explore the cluster and check local and remoteQMS for the best resource matching. In fact, when a query explores a QMS and related resources, it shows one of the following two behaviors: First, a query does not find the requested resource, then it goes to the next QMS, and continues until the best matching for discovery is achieved or the query expires. If a query that is exploring a QMS associated with aggregate-node a is not satisfied with the local resources it goes to the next aggregate-node b in one of the QMS neighbors with a given probability, which is proportionally related to the rate of Pf_{ab}^r for each one of r resources over edge $ed(a, b)$ (See Equation 4.1).

$$Probability_{ab}^r = \frac{Pf_{ab}^r}{\sum_z Pf_{az}^r} \quad (4.1)$$

z is an aggregate-node which is in vicinity of a . And r is a resource type which probably existed in the QMS of b .

According to Equation 4.1, queries use the resulting probability to select the next QMS

for discovery. After finding a match for a query, the reply message includes the description of the qualified discovered resources which will be returned to the initial requester. Among all the potential resources the best matching is a resource with lowest rate of latency. In the next step the probe factor for the result of the performed discovery procedure would be updated according to Equation 4.2.

$$Pf_{ab}^r = \begin{cases} Pf_{ab}^r - \delta Pf_{ab}^r + \frac{\delta}{L_{ab}} & \text{if } r \in QMS_b \\ \delta Pf_{ab}^r & \text{if } r \notin QMS_b \end{cases} \quad (4.2)$$

δ is a random number which is $0 < \delta \leq 1$ and L_{ab} is the latency between a and b . r is the requested resource with type t . The default value of Pf is 1.

4.6.3 AN Algorithm Description

RP_{ans} (i.e., vnodes with the module-role of AN which provide Query Management Service) operate as following:

RPs assign several different event-handlers to manage the communication events. They also receive and inspect each incoming message to identify the event that must be triggered. Depending on the message type in the client-side (source event generator) and the vnode type (i.e., node type or module-role) in the server side (target event handler) various operations (i.e., event handler functions) might be performed by receivers. Below we elaborate on the event receivers' behavior (event-handling functions) for different events when the node type of the receiver node is set to AN.

The **Lookup** event handling in the RP_{ans} is similar to **Lookup** event handling at the RP_{lns} . The only difference is that the AN receiver nodes play the role of a LN.

The RP_{ans} are the major targets for the main-query events. Whenever such event occurs in the receiver side, that is supposed to be the local QMS of the requester, several tasks are consequently performed by different sub-components of the QMS (e.g., Query-Analyzer, Query Router (QROUT), Query Registry (QREG), etc) (see Figure 4.19). At first, the main-query is registered in the QREG. This specifies the current QMS provider, as the main responsible entity to collect and manage the overall results of the main-query. Afterwards the Query-Analyzer splits the main-query in multiple sub-queries (i.e., queries) based on the main-query template and the homogeneity of resources in each sub-query. These sub-queries in turn update their query-routing information in the QROUT. Depending on the sub-query conditions for the different layers, the Query-Analyzer makes a decision for each individual and independent sub-query based on their query-schemes, focused entirely towards conducting the best possible exploration pathway around the system.

For the sake of simplicity, in the rest of this paper, we use a query-scheme to represent sub-query conditions in different layers (the number of homogeneous resources required for each sub-query is a separated query argument which is not shown in the query-scheme). A query-scheme represents all sub-query conditions in 3 layers as $\langle c_{ln}, c_{an}, c_{sn} \rangle$. Here, c_{ln} , c_{an} and c_{sn} denote the existence of query constraints for the layer_{ln}, layer_{an} and layer_{sn} accordingly, while n_{ln} , n_{an} and n_{sn} determine non-existence of any query-conditions on those layers.

The strategy to split a query by Query-Analyzer is related with the description of the query. The Query-Analyzer simply splits the query (a heterogeneous group) to multiple sub-queries (multiple homogeneous groups), as it is originally described by the query description (see

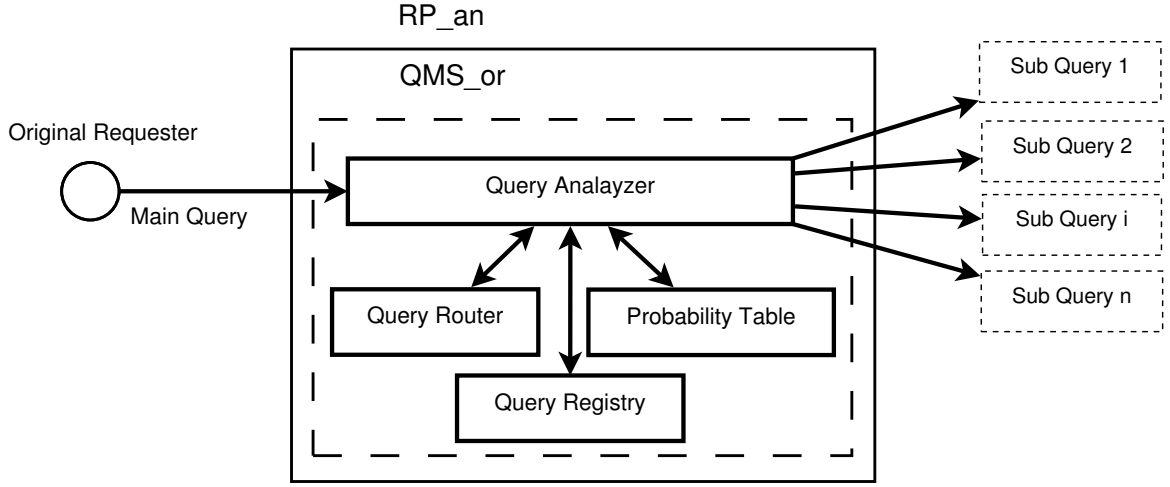


Figure 4.19: Main query processing in QMS_{or} .

Section 3.4.2). For example, for the query description presented in Listing 3.4, the Query-Analyzer can only create a single sub-query, since the query description only contains one homogeneous group (HOGroup1). The resulting sub-query aims to find 5 similar processing cores with the characteristics described by SingleNode1. As it is shown in Listing 3.4, the sub-query does not provide any condition in Layer3 ($\langle c_{ln}.c_{an}.n_{sn} \rangle$). Similarly, in other example for the query described in Listing 3.5, the Query-Analyzer splits the query into two sub-queries (sub-query1 and sub-query2), since the query description contains two homogeneous groups (HOGroup1 and HOGroup2). The sub-query1 is similar to the one discussed for Listing 3.4. The sub-query2 aims to find a group of similar resources, including 8 processing cores with attributes described by SingleNode2. The query-scheme for sub-query2 can be presented as $\langle c_{ln}.c_{an}.c_{sn} \rangle$, since it provides conditions in all layers.

The conduction of sub-queries by the Query-Analyzer can be done as following: (a) for the sub-queries with the query-scheme $\langle c_{ln}.n_{an}.n_{sn} \rangle$, the RP_{an} initiates a DHT lookup search (i.e., searching in the local mini-cell) by sending a DHT Lookup message to the entry node in the DHT ring. (b) if the query-scheme is $\langle c_{ln}.c_{an}.n_{sn} \rangle$, the RP_{an} first checks if its aggregate-stamp is validated according to the aggregate-key of the query. In other words, it determines whether the $layer_{an}$ information of the local QMS fulfills the sub-query conditions in this layer or not. Thus, if the $layer_{an}$ information of the current node is validated by the c_{an} query conditions, the RP_{an} continues with the DHT lookup, otherwise, the RP_{an} , leveraging the probability table, selects the next QMS (i.e., another RP_{an} in the current layer which represents a mini-cell with the higher probability to find the requested resources) and sends a **Downward** discovery message towards the QMS address of the next mini-cell. (c) if the query-scheme is $\langle c_{ln}.c_{an}.c_{sn} \rangle$, the RP_{an} generates and sends a **Forward** discovery message towards the higher layer which offers the SQMS service (the RP_{an} relays the query to its SQMS address in the super-node layer).

Figure 4.20 demonstrates examples of query processing for sub-queries with different schemes and due to the decisions made by the Query-Analyzer. Upon receiving a request from a HARD user, the RR entity starts the discovery procedure by sending a **Main-Query** to its local pre-assigned QMS provider (QMS_{or}). The **Main-Query** includes only a single sub-query.

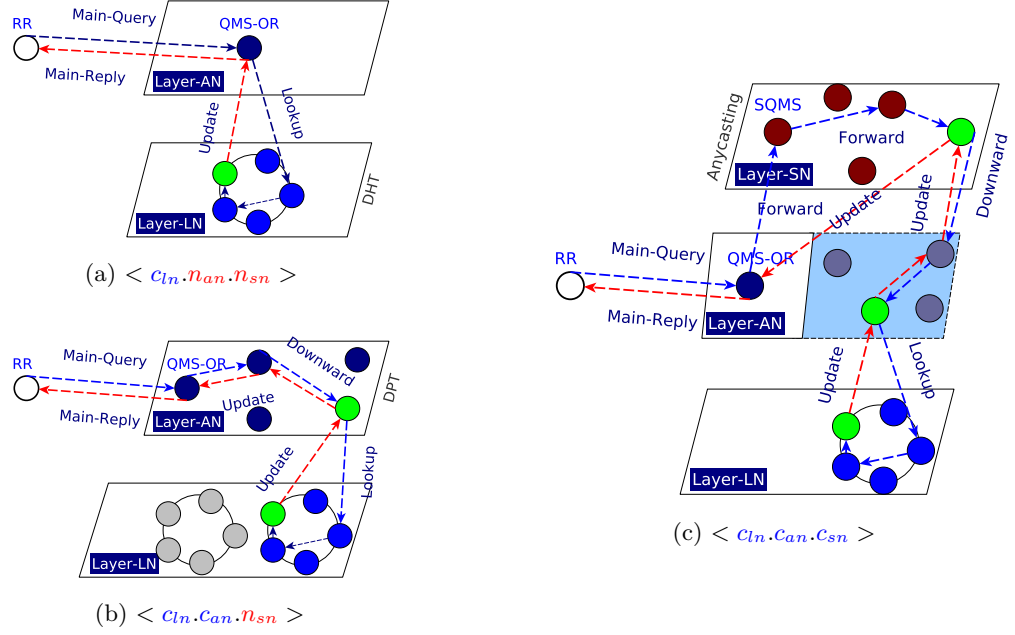


Figure 4.20: Examples of sequence of the resource discovery message flow for different sub-query schemes (the **Main-Query** contains a single sub-query).

In Figure 4.20-a, the sub-query scheme is $\langle c_{ln}.n_{an}.n_{sn} \rangle$, which means that the query only introduces conditions for the $layer_{ln}$. Thus, the query is delivered to the lower layer ($layer_{ln}$) with a **Lookup** event. This will result in a DHT lookup. The query successfully returned a hit by sending an **Update** message to the QMS_{or} . The QMS_{or} finally sends the overall query results to the requester by using a **Main-Reply** message. In Figure 4.20-b, the sub-query scheme is $\langle c_{ln}.c_{an}.n_{sn} \rangle$. Therefore, the query must find a match for its c_{an} conditions, before going to the lower layer. This can be done by dispatching the query within $layer_{an}$ and visiting potential ANs, by using probability tables and **Downward** messages. The query finally finds a matched AN and then sends a **Lookup** message to its lower layer in order to process the rest of the query conditions, similar to the first example. Figure 4.20-c depicts similar processes for a sub-query with query-scheme $\langle c_{ln}.n_{an}.c_{sn} \rangle$. Due to the c_{sn} constraints, the discovery must be initially started from the upper layer ($layer_{sn}$). A **Forward** message is used to transmit the query to the top layer. Subsequently, one or multiple anycasting might be required to find a matched SN. The sub-query then transferred to the $layer_{an}$ for the rest of discovery process in the lower layers.

HARD3 defines four different types of **Update** events, which are elaborated in Table 4.4. The **Update** event handler in a RP_{an} generally performs the following operations in response to the occurrence:

(a) Depending on the content and the type of the incoming **Update** message, RP_{an} updates the values of the relevant fields in its local probability table. As it is detailed in Algorithm 6, Updating is required only for **Update_{sys}** and **Update_{vic}** events. On a **Update_{vic}** event, the updating process is terminated in this step since the purpose of this event is just to update the resource knowledge of neighboring aggregate-nodes of a potential SoR (the event generator). On **Update_{sys}** and **Update_{nul}** events, the receiver realizes that the discovery process for the given sub-query is completed, thus, the **Update** messages follows the reverse path taken by

the query message to the QMSor (i.e., the QMS which has initially registered the query) through the overlay network. The only difference between these two events is that `Updatesys` is required to update the probability table of the intermediate nodes but `Updatenul` does not need to do this.

Algorithm 6: DPT Updating

```

Input: sq= sub-query, up=updater, ut=update-type or type of sub-query
/* sq:sub-query, rt:resource-type, nr:neighbor-record,  $\varphi()$ :quality fun(), sor:source-of-resource */
sorpf(sq, up, ut, sornew, sorold, pfold, flag)
    // a function to calculate the new pf value and making decision for the new sor
    result.sor=-1 // indicates that the sor value should not change
     $\delta$ =random(0,1) //  $0 < \delta \leq 1$ ,  $\lambda$ :latency */
     $\lambda$ =sq.receiver-time - sq.sender-time //  $\lambda$  is the latency by ms between the sq-sender or updater
    and the sq-receiver
    if ut==Updatesys then // or if sq.type==Updatesys
        if (sq is fully resolved) $\wedge$ ( $\exists$  sornew) then
            result.pf=pfold- $\delta * (pf_{old}) + \frac{\delta}{\lambda}$  // potential increase in pf value
            if  $\varphi(sor_{new}) > \varphi(sor_{old})$  then
                result.sor=1 // a new sor, suggested by sq, can replace the previous sor in the
                DPT
            else
                result.pf= $\delta * (pf_{old})$  // potential reduction in pf value
                if (sorold==sornew) $\vee$ (flag==0) then
                    result.sor=0 // the previous sor in the DPT must be removed by setting the sor
                    value to empty
        else if ut==Updatevic then
            result.pf=pfold- $\delta * (pf_{old}) + \frac{\delta}{\lambda}$ 
        else if ut==Downwardcbk then
            result.pf= $\delta * (pf_{old})$ 
        return result

    sornew=sq.preferred-vnode // getting the preferred-vnode/SoR of sq
    foreach rt : rtid  $\in$  sq.res-type-ids do
        if DPT.find(rt) then // rt already exists in the DPT
            RTrecord=DPT(rt).get(); flag=1
        else // rt does not exist in the DPT
            DPT.add(rt); RTrecord.sor.set-empty(); RTrecord(nb).pf=1.0; flag=0
        foreach nb  $\in$  List of AN Neighbors do
            if nb==up then
                sorold=RTrecord.sor
                pfold=RTrecord(nb).pf
                res=sorpf(sq,up,ut,sornew,sorold,pfold,flag)
                RTrecord(nb).pf=res.pf
                RTrecord(nb).probability= $\frac{RTrecord(nb).pf}{\sum_{nr \in ALLNeighbors} RTrecord(nr).pf}$ 
                if res.sor==0 then
                    RTrecord.sor.set-empty() // SoR sets to empty
                else if res.sor==1 then
                    RTrecord.sor=sornew // SoR changes
                else
                    RTrecord.sor=sorold // SoR does not change
                DPT(rt).set(RTrecord)
    return

```

(b) On `Updateqms`, the receiver (which is a *RP_{an}*) investigates the current status of the query considering the discovery results as provided by the received `Update` message. Accordingly, three different possibilities would be considered: the query is fully resolved, the query is partially resolved and the query is not resolved. When the query (i.e., the sub query which has already registered in the QMS of the original requester) is fully resolved, a corresponding `Updatesys` event is created and sent back to the QMS_{or} while the probability tables and the

QROUTs of the revisited nodes will be updated. In fact, due to the temporal knowledge of the Query-Routers in each node, the Update_{sys} message backtracks its way to the QMS_{or} by traversing the nodes in different layers.

(c) In other case, when a sub query is partially resolved, RP_{an} properly reshapes the sub query as a **Downward** message including information on the partial discovered resources in order to continue the search to find the rest of the requested resources within the current layer. Probability tables assist RP_{ans} to efficiently decide, choice of the next QMS destination. There are two search ending conditions in layer_{an} : the first one is reached if all the potential QMS providers in the current layer of the current cell are explored, thus, an **Upward** message including the partial results (if there is any) will be transferred to the address of the SQMS provider in the upper layer. A potential QMS provider is an aggregate-node, representing a group of leaf-nodes in a mini-cell, which the probability to find the desired resources for a sub-query among its subsidiary resources is more than a specific threshold value. We must note that, even among the potential QMS providers, only qualified QMS providers which fulfills the c_{an} conditions for a given query will be deeply searched in layer_{ln} level; the second layer_{an} search ending condition is also reached if the Time To Live (TTL) (i.e time to live in terms of number of hops or expiration time) of the sub-query is expired, therefore, an Update_{nul} will be sent to the QMS_{or} .

(d) Upon receiving an Update_{sys} or an Update_{nul} message by a QMS_{or} (i.e., the main query registry point), the local QREG of the QMS_{or} will be updated according to the discovered results of the sub query and if all the other sub queries of the main query are also resolved or completed, a final main-reply message including the results of all the sub-queries will be sent to the original requester (the issuer of the given main-query) (see Algorithm 7).

As we elaborated in Table 4.4, there are two types of **Downward** events: normal **Downward** and call back **Downward**. A **Downward** event is triggered when the query conditions in the upper layer are met. It is the major communication event which is happening in layer_{an} , and notifies the receiver that the required number of resources are not fully discovered yet. The query transmission in layer_{an} is performed by employing DPTs, and exploits the query status information from the content of the received query at each aggregate-node to manage the relying process. The query status information contains the fields such as the query-scheme (i.e., the query conditions in each layer), inter resources communication constraints, number of requested resources, number of discovered resources, resource-IDs for the discovered resources, preferred-vnode, source address, destination address, etc. Whenever a **Downward** query is received/sent from/to an/a aggregate-node/super-node, the query routing information (such as main-query-id, sub-query-id, parent-sender and destination) in the QROUT must be updated. Furthermore, for a given query, the corresponding information in the QROUT of the revisiting nodes automatically will be removed before generating the events such as Update_{sys} , Update_{nul} , **Upward** and Downward_{cbk} . In fact QROUT only traces the mainstream of the queries which contain the events such as **Downward**, **Upward** and **Forward**.

The search in layer_{an} is conducted over a tree graph where the tree's root is the first aggregate-node in the current layer which receives the **Downward** query from the other layers (i.e., upper or lower layers). The QMS in each aggregate-node makes a decision to relay the query to one of the neighboring aggregate-nodes if it is required. This is performed depending on several parameters such as the query's resource-type-ids, the query's preferred-vnode and the probability to find the required resource type in the path which is specified by a neighbor-node as the next QMS destination. The query's resource-type-ids denote the IDs for different resource types due to query dimensions. Each query dimension specifies a desired value or

Algorithm 7: Processing an Update_{qms} sub-query by a RP_{an}

```
Input: sq= The received  $\text{Update}_{qms}$  message
/* sq, temp:sub-query, qrg:QREG, qrt:QROUT, QMS: the QMS provided by  $\text{RP}_{an}$  */
while sq.TTL is valid do // if sq is valid due to its Time to Live
    if sq is fully resolved then // if all resources required by sq has already been discovered
        if sq is registered in QMS.qrg then // if QMS is the  $\text{QMS}_{or}$  of sq
            qrg.addDiscoveryResults(sq) // adding results of sq to overall results collected by
            other sub-queries of the main-query
            if qrg(sq.mainQID).isCompleted then // checking if all sub-queries of the main query
            have already been resolved
                send(Main-Reply,qrg(sq.mainQID).sender)
                /* sending the final Main-Reply of the query to the original requester */
        else // if QMS is not the  $\text{QMS}_{or}$  of sq
            send( $\text{Update}_{sys}$ , qrt(sq.subQID).sender); // sending an  $\text{Update}_{sys}$  to the sender of sq (the
            last visited  $\text{RP}_{an}$  by sq) through using sub-query tracking information recorded in
            QROUT
            if the current vnode is the preferred-vnode for sq then
                broadcast( $\text{Update}_{vic}$ , all members of AN-Neighbors)
                // sending an  $\text{Update}_{vic}$  message to all neighbors of current  $\text{RP}_{an}$ 
            qrt(sq.subQID).remove // removing sq tracking info from QROUT
    else // if sq is not resolved or partially resolved
        temp=sq; temp.nRR= sq.nRR- sq.nDR; temp.nDR=0 // adjusting/reshaping sq (as sub-query
        temp) based on current discovered resources and the remaining resources required, the
        sub-query temp maintains list of all previously discovered resources by sq and it is
        identical to sq except for the changes
        temp.type=Downward|Downwardcbk |Upward // QMS makes decision for the type of sub-query
        temp, the initial choice is Downward, if it is not possible, then Downwardcbk is the
        choice, and in the case that all ANs in the current cell have already been searched,
        Upward is the choice
        qrt(sq.subQID).update
        Continuation of the search with the modified sub-query temp // since sq is not fully resolved
        yet, the search must be proceed either in the current layer (layeran using Downward or
        Downwardcbk) or in the upper layer (layersn using Upward)
    if sq.TTL is expired then
        if sq is registered in QMS.qrg then
            send(Main-Reply,qrg(sq.mainQID).sender)
            qrg.dreg(sq.mainQID) // removing main query information (including sub-queries result
            info) from QREG
            removeQroutInfo(sq.mainQID) // removing main query information (including sub-queries
            routing info) from QROUT
        else
            The sq will be destroyed;
            The sq routing info in the QROUT of all visited nodes (by sq) will be removed;
            The  $\text{QMS}_{or}$  will be notified to issue the final Main-Reply for the query;
return
```

range of values for a single resource attribute. The query-analyzer (i.e., a QMS component) in the local QMS of each main-query is responsible for recognizing the list of resource-type-ids for each created sub-query. On the other side, each QMS in the system constructs, maintains and updates one small probability table containing fields such as neighbor-id, probe-factor(pf) and probability for each demanded resource-type. Moreover, for each resource-type a preferred source of resource might be assigned or modified during the update procedure, which represents the current QMS's preference to direct the related queries to a SoR that provides quality, in terms of size (i.e., number of available resources in SoR), and distance (i.e., latency or number of hops between current QMS and SoR).

Figure 4.21 depicts a simple example of DPT in a system with 2 predefined attributes (CCR: Core Clock Rate, L1S: L1 Cache Size) over 4 different ranges of values (resource-types 1-4). We must note that the number of predefined resource-types in the system is an arbitrary

design choice. Also the number and definition of resource-types must be uniform among all probability tables in the system. Defining a large number of resource-types in the system may increase the resolution of DPTs (i.e., accuracy of probability tables). However, it might have memory cost, resulting DPTs with larger sizes (HARD limits the number of resource-type definitions for each single attribute in the range of 2 to 4). In our example, a requester (RR) sends a **Main-Query** to its local QMS provider (*vnode-11*). The resources desired for the query include two different processing cores (core-A and core-B): 5 cores with CCR=1500 MHz and L1S=512 MB and 8 cores with CCR=2200 MHz and L1S=256 MB. The query is simple and does not introduce any conditions in $layer_{an}$ and $layer_{sn}$ (i.e., $\langle c1_{ln}.n_{an}.n_{sn} \rangle$ and $\langle c2_{ln}.n_{an}.n_{sn} \rangle$). The Query-Analyzer in the *vnode-11* splits the **Main-Query** into two sub-queries, bearing in mind that each sub-query must represent query conditions for required resources, specified by the query description, which are identical to each other (i.e., resources required by a sub-query are a group of homogeneous resources). In other words, sub-query1 aims to find 5 processing cores of type core-A and sub-query2 is going to find 8 processing cores of type core-B. These two sub-queries finally return their results achieved to the QMS_{OR} in *vnode-11*. The QMS_{OR} aggregates the results and sends a **Main-Reply** to the requester (*vnode-7*).

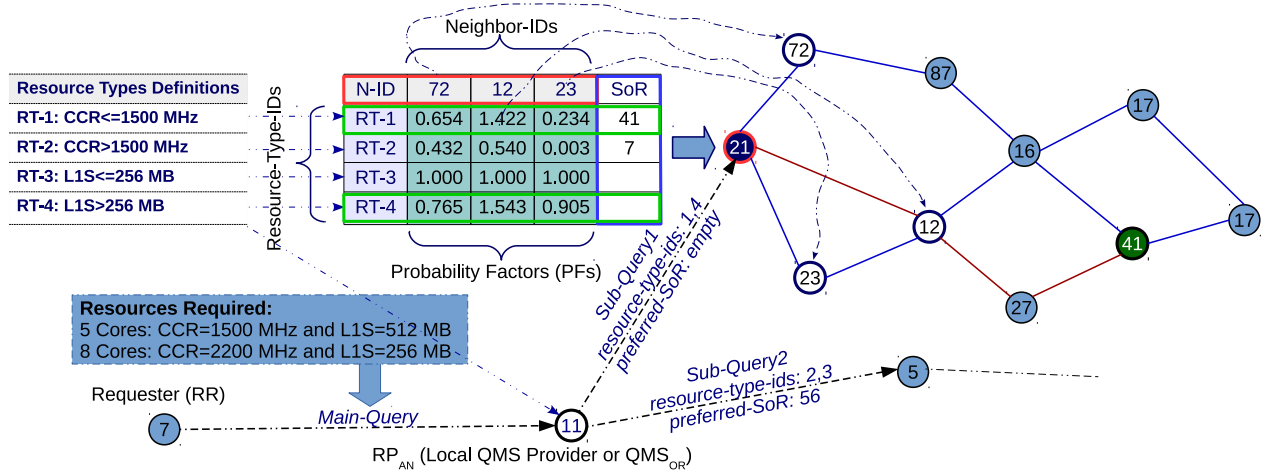


Figure 4.21: An example of DPT in a system with 2 predefined attributes over 4 different ranges of values (resource-types or resource-categories) (CCR: Core Clock Rate, L1S: L1 Cache Size).

Figure 4.21 shows that sub-query1, in continuation of its exploration path, arrives in *vnode-21*. Due to the querying policies for the current layer, if the sub-query is required to be transferred into another QMS provider in the neighborhood, therefore one of the neighboring *vnodes* must be selected as the next QMS destination. This can be done by using the probability table in *vnode-21*. This table includes the list of direct neighbors (*vnodes-72*, 12 and 23) as well as the probe factor for each neighbor for each resource-type-id. The sub-query1 contains its list of desired resource-type-ids (1 and 4) which have been already specified by the Query-Analyzer of the QMS_{OR} of the sub-query. For the sub-query1, the QMS in *vnode-21* only analyzes the resource-type records RT-1 and RT-2 which are matched with the resource-type-ids of the sub-query. Accordingly, a neighboring *vnode* with the

absolute maximum of probability in both records (*vn*-12) will be selected as the sub-query destination. Each resource-type record also includes a SoR suggestion. The sub-query1 sets its preferred-vnode to 41, since its original preferred-vnode is empty and also both records RT-1 and RT-4 collectively propose an absolute value for SoR. We must note that values for PFs and SoRs in the probability tables are dynamically updated due to the querying data provided by each transacted query (the initial values for PFs are 1, as they are for RT-3, and the SoR value might be empty, as it is for RT-3 and RT-4).

The type of sub-query1 in the above example (Figure 4.21) is **Downward**. In fact, upon receiving a **Downward** query by a QMS, if the query conditions in $layer_{an}$ are met by the aggregate-stamp, a **Lookup** query is conducted to search in the current mini-cell, otherwise the QMS, using its probability table, must select the next **Downward** destination to continue the search.

In order to select the next QMS provider among the neighboring aggregate-nodes, as it is presented in Algorithm 8, at first, the QMS checks if there exist any absolute SoR preference either by query itself or by QMS (i.e., query based or QMS-based preference). An absolute preference only exists if an absolute maximum number of the probability tables corresponding to the resource types mentioned in the resource-type-ids of the given query agree to vote to a specific SoR.

Algorithm 8: Selection of the next QMS provider

```

Input: sq= sub-query
/* sq:sub-query, rt:resource-type, nb:neighbor, vn:vnode, sor:source-of-resource,
   op:overall-probability */
const: call-back /* indicates that a Downwardcbk must be initiated, an Upward is initiated when it is
                  not possible to initiate a Downwardcbk */
∅:vector, temp:vector-record // overall probability for the neighbors
A={nb : nb ∈ AN − Neighbors}; B={vn : vn ∈ sq.visited − qms − ids}
C=(A ∩ B') // A is the list of AN neighbors, B is the list of all already visited ANs by sq and C
             is the list of neighbors which have not yet been visited by sq
if |C|==0 then return call-back
if (there is sq.preferred-vnode) ∧ (sq.preferred-vnode ∈ C) then
    return sq.preferred-vnode
foreach rt : rtid ∈ sq.res-type-ids do // calculating the average probability for each of the C members
    with respect to different resource-type-id specified by sq.res-type-ids

    if DPT.find(rt) then
        RTrecord=DPT(rt).get()
        foreach nb ∈ C do
            if ∅.find(nb) then
                ∅(nb).op=mean(∅(nb).op, RTrecord(nb).probability)
            else
                temp.neighbor=nb
                temp.op=RTrecord(nb).probability
                ∅.add(temp)

if ∅.size()==0 then return random(nb : nb ∈ C)
D={nb : (∅.find(nb)) ∧ (∅(nb).op == ∅.Maximum − op) ∧ (∅(nb).op > 0)} // D is the list of C members
with the maximum overall probabilities
if |D|==0 then return random(nb : nb ∈ C) // returning a random member of C as the next QMS destination
else return random(nb : nb ∈ D) // returning a random member of D as the next QMS destination

```

HARD3 introduces two different mechanisms for SoR tracing: query-based preference and QMS-based preference. In the query-based preference, the query obtains the value for its preferred-vnode (note that each query message includes the information of the preferred-vnode

of the query) based on the QMS's SoR preference of the QMS_{or} and it keeps relying on this fixed preference until discovery ends. But in the QMS-based preference, the query in each intermediate QMS dynamically applies to use the SoR preference of the current QMS. On the existence of either the preferred-vnode for the query (in query-based method) or the absolute SoR preference for the current QMS (in QMS-based method), RP_{an} gives priority to a neighboring QMS provider, which is preferred by SoR preference or preferred-vnode as the next query destination. However, if the query doesn't have a target preference, a neighbor, with the highest probability among all neighbors is selected. In the case that the number of neighbors with highest probability is higher than one, a random destination is selected among the highest ranked neighbors. It must also be taken into account that, the overall probability to select a neighbor as destination for a query in a QMS, is measured by averaging the probability values provided by the local probability tables, corresponding to the resource types, identified in the list of resource-type-ids of the query.

Whenever a RP_{an} ensures that all the potential QMS providers in the vicinity of the current node are explored, and since the query is not completed, a **Downward_{cbk}** message must be sent to the sender of the original query in the upper level of the search tree (i.e. the query parent), using the query tracking information recorded in the QROUT. The query itself also keeps the information about the visited nodes, which helps the query to efficiently explore the tree graph. By triggering the **Downward_{cbk}** event in the receiver-side, the normal **Downward** procedure is performed, exploring other potential branches of the tree, by directing the query to a qualified unvisited neighbor, while the probability tables in the **Downward_{cbk}** receiver are updated due to the query's characteristics and search results, as we elaborated in Algorithm 6. If consecutive down-warding processes (i.e., series of **Downward_{cbk}** and **Downward**) fail to complete the given query in different levels, and branches of the search tree and the **Downward_{cbk}** finally reach the entry QMS provider (i.e., the first QMS provider in the current layer of the current cell which initiated the **Downward** query within this layer), a RP_{an} sends an **Upward** message to its SQMS address in the upper layer (i.e., layer_{sn}), informing the SQMS provider that search must be continued in other qualified cells in the distributed environment.

4.6.4 Anycast-based Discovery

Anycast can be considered as a powerful paradigm for resource discovery in large scale distributed systems. It enables communication between a source node and the nearest (or the best) member of an anycast group. The proximity metric (or the metric for being the best) can be defined in terms of hop count, delay or the minimum amount of load [410]. However, at present, there are challenging issues that prevent easy adoption and deterministic deployment of anycasting in general, and particularly for reliable and scalable resource discovery applications. These limitations include the issues such as followings:

- Challenge (CH)#1: Lack of support for session-oriented communications which makes anycasting inappropriate specifically for stateful applications [411].
- CH#2: Lack of support for globally scalable anycast routing in the current network routers, due to the fact that routing paths to anycast groups cannot be aggregated [287, 412].
- CH#3: Dynamic nature of anycast creates a significant overhead for updating and maintenance of the anycast groups, specifically for discovering resources in large scale

dynamic distributed computing environments (i.e., including joining, leaving or failing nodes/resources) [287, 412].

We propose a novel specification-based anycasting scheme to resolve queries in layer_{sn} while dealing with the aforementioned challenges. Our approach uses a shortest path, routing-based anycast method, which uses the top level layer stamps (i.e., node's specifications in SN layer) to create anycast groups, while nodes in this layer are able to dynamically adjust their own anycast group based on the required resource conditions, provided by incoming discovery requests from the lower layers.

CH#1 denotes that stateful communications cannot be directly supported by native network-level anycasting, since there is no guarantee that subsequent packets of the same session arrive at the same anycast server. The reason is that the routing path between certain nodes (e.g., the source and destination node) may vary with changes of the network configuration, and consequently, anycast packets may arrive at different members of the anycast group. Our resource discovery approach does not suffer from the issue mentioned in CH#1, since it does not rely on stateful communications. Instead, a discovery requester (i.e., a SN) initially sends a **Forward** request (using anycast) to find the closest SN in the anycast group. Once an anycast member receives a **Forward** request it can explicitly communicate with the original requester using the sender's unicast address. Hence, the requester would be able to perform the rest of the transaction using traditional unicast operations.

In order to deal with CH#2, we propose two different approaches for anycasting, which are as follows:

Network-level Approach: We extend the functionality of the routers, so that they are able to distinguish between regular unicast routing requests and anycast routing requests. However this modification is bound to SN routers (i.e., the routers that directly communicate with SN nodes). By doing this, it would be necessary to upgrade the regular routers. Both type of routers (SN routers and non-SN routers) also operate as regular routers for non-anycast (unicast) requests. Using this approach, upon receiving a discovery request (from the requesters in the lower layers of the current hierarchy or from other RP_{sn} in the same layer) at a RP_{sn}, the receiver would be able to forward the request to the closest RP_{sn}, by extracting the anycast address of the destination from c_{sn} (i.e., the description of the query conditions in layer_{sn}, which is the representative of the specifications of the desired resources in SN-layer) if the c_{sn} requirements failed to be met in the current SN. The discovery request would be transferred to the lower layers of the current hierarchy whenever the c_{sn} requirements are fully obtained by SN node in top of the hierarchy.

Application-level Approach: Each SN creates its own anycast address by hashing the layer-stamp of the SN node, which is representative of all the specifications of the resources in layer_{sn}. Using this approach, the anycast address of each SN is a function of the resource specifications in the super-node's layer. The anycast address of a node may change when the node's specifications (e.g., dynamic resource attributes) are modified. SN nodes with uniform specifications advertise the same anycast addresses. The creation and maintenance of the anycast groups are significantly lightweight, since the anycast groups can automatically be created and they can also dynamically be changed without requiring communication among the group members or group registration. SN nodes advertise their anycast address, as well as unicast address, to other SN nodes in their neighborhood. The neighborhood can be specified based on proximity metrics such as number of hops or delay. Whenever a RP_{sn} decides to anycast a discovery request to other potential SNs in the network, it passes the request to

its local anycast-resolver, which is responsible for determining the unicast address of the best possible destination. Subsequently, the discovery request would be forwarded to the destination by using its unicast address and regular routing. The anycast-resolver is a SQMS sub-component which generates the anycast address of the query destination, using the given query conditions, and maps the resulted anycast address to a specific unicast address of the destination. It operates as following: The anycast address of the potential destination for a given discovery request, is extracted from the c_{sn} of the request. The anycast-resolver checks the SNs registered in the current node and selects the ones which anycast address is similar to the anycast address of the request. In the next step, the unicast address of a SN with less distance (in terms of latency or number of hops) is returned to RP_{sn} as the final destination of the request. If the desired anycast destination is not found among the (list of) registered SNs, the discovery request is forwarded to the closest SN in the list. If the list is empty, the query is terminated and the proper **Update** (i.e., reply) message is sent to the requester. If c_{sn} conditions for the given discovery request does not exist (i.e., when the creation of the anycast address is not feasible), the discovery request is forwarded to the closest SN.

Whenever a RP_{sn} receives a query, a query type retrieval is performed, and different mechanisms and procedures are conducted to resolve or redirect the query. The **Forward** event is the most regular communication event within layer_{sn}. A **Forward** receiver (a node which receives a **Forward** message), first assesses whether the layer stamp (i.e., the super-node-stamp) will be qualified due to the c_{sn} query conditions or not. If it is qualified, a **Downward** self-event is triggered in the current super-node where the SQMS provider, as the event-receiver acts as a QMS provider, which conducts the **Downward** query to lower layers. The SQMS provider, later, will receive a response from the lower layers either in the form of **Upward** (if the query fails or remains uncompleted in the lower layers) or **Update** messages (if the query is resolved or completed). On the occurrence of a **Update** event, the RP_{sns} redirect the incoming messages to the original requesters. Receiving a **Upward** event is exclusively dedicated to RP_{sns} (see Table 4.4 and Figure 4.15). In fact, on occurrence of an **Upward** event, the RP_{sn} will know that the requirements of the correspondent query can not be met in the lower layers of the current SN's cell, and the query must be directed to other remote cells in the system.

4.6.5 SN Algorithm Description

The RP_{sns} are the resource providers which provide SQMS services to entities in their local cell, and to other SQMS providers in the system. Depending on the type and status of the event triggered and the characteristics of the given query, SQMS providers might behave as RP_{sn} , RP_{an} or RP_{lns} .

As depicted in Algorithm 9, if a RP_{sn} receives an **Upward** query, it means that the receiver already has been qualified for the c_{sn} query conditions, but the query conditions in the lower layers have not been achieved. Accordingly, the RP_{sn} sends a **Forward** message to an anycast address extracted from c_{sn} of the given query. The **Forward** message will be automatically redirected to the nearest SQMS provider in the system, which has the same anycast address. Using the proposed anycast scheme significantly reduces the search space for the given query. It automatically limits the search space to only the SQMS providers in the system that certainly would be able to fulfill the c_{sn} query conditions. However the query conditions in the lower layers must be examined in each target forward-receiver separately.

Algorithm 9: Anycast based Forwarding in RP_{sn}

```
Input: sq= The received sub-query message
/* sq:sub-query, nb:neighbor */
if sq.type==Forward then
    if  $RP_{sn}$  is qualified according to sq.csn then
        send(sq, Downward,  $RP_{sn}$ )
        /* generating a Downward self-event by sending a self-message to  $RP_{sn}$ , as a result,  $RP_{sn}$ 
           behaves as a  $RP_{an}$ , receiving a Downward message */
    else
        anycast-address=mapped-ac-address(sq.csn) // generating an anycast address by mapping the
        specification of the sub-query sq in the layersn into an anycast address
        anycast(sq,Forward, anycast-address) // anycasting the sub-query sq with the type Forward to
        the extracted anycast address
    else if sq.type==Upward then
        if there is sq.csn then
            anycast-address=mapped-ac-address( $RP_{sn}$ .layer-stampsn) // generating an anycast address by
            mapping the specification of the  $RP_{sn}$  in the layersn into an anycast address
            anycast(sq,Forward, anycast-address)
        else
            target=random(nb:(nb ∈ SN-Neighbors)^(nb ∉ sq.visited-qms-ids) ) // selecting a random
            neighbor from the list of SN neighbors which have not yet been visited by sq
            send(Forward, target)
    else if sq.type==Update then
        |  $RP_{sn}$  performs the corresponding update procedure similar to  $RP_{an}$ 
```

4.6.6 Multi-dimensional Discovery

The resource description is not supposed to only support single attribute based discovery, rather it should be able to be used for multidimensional range queries. Therefore as a multidimensional abstraction in a system in which the information of resources are distributed in hierarchical layers, we assume our abstraction system to have L layers l_1, l_2, \dots, l_L , and each resource to have A attributes a_1, a_2, \dots, a_A . Then according to the layers description each attribute a_i is placed in the layer l_j . This is depicted in Equation 4.3.

$$a_i \in l_j \text{ where } 1 \leq i \leq A \text{ and } 1 \leq j \leq L \quad (4.3)$$

Each attribute is described by a triple $\langle At_ID : Op_ID : At_VA : La_ID : At_TY \rangle$, which parameters are Attribute ID (At_ID), Operator ID (Op_ID), Attribute Value (At_VA), Layer ID (La_ID) and Attribute Type (At_TY : String or Number). La_ID can be extracted from At_ID and we assume bitmap and numeric values to represent the string types, so that an attribute can be defined in a triple format like $\langle At_ID : Op_ID : At_VA \rangle$, where At_ID and At_VA are numerical values and $OP = \{<, \leq, =, >, \geq\}$. Using a hash function H , we create $H(At_VA)$ for every At_VA values, besides, in order to support range querying the hash function has to preserve locality. It means that for each attribute a_i :

$$\text{if } At_VA(a_i) \in [VA_{min}, VA_{max}] \Rightarrow H(At_VA(a_i)) \in [H(VA_{min}), H(VA_{max})] \quad (4.4)$$

Our proposed resource discovery solution supports four different approaches for range-based multidimensional discovery.

Approach A: Each resource has to generate hash values for its information in all dimensions. Afterwards it uses a priority function to place hashed values in an ordered list PL . In the next step, each resource will create a uniform key of H_T which demonstrates all of the

resource properties in a specific layer. So, for each resource r_i we have:

$$H_T(r_i) = f(H(At_{VA}(a_1)), H(At_{VA}(a_2)), \dots, H(At_{VA}(a_A)), PL) \quad (4.5)$$

Finally each resource r_i has to register its uniform key in $S(r_i)$, where:

$$S(r_i) = successor(H_T(r_i)) \quad (4.6)$$

Approach B: We store resource information for each dimension in an individual DHT ring. Therefore each resource r_i with A attributes collaborates in D particular DHTs where $A = D$. For example in order to perform a range query in such environment, we will be looking for resources with set of specific attributes a_1, a_2, \dots, a_A in the layer of l_j which attribute values $At_{VA}(a_1), At_{VA}(a_2), \dots, At_{VA}(a_A)$ should be in following ranges respectively:

$$[VA(a_1)_{min}, VA(a_1)_{max}], [VA(a_2)_{min}, VA(a_2)_{max}], \dots, [VA(a_A)_{min}, VA(a_A)_{max}] \quad (4.7)$$

We distribute the resource information within layer l_j to D distributed hash tables H_1, H_2, \dots, H_D where each dimension is stored in a DHT:

$$H_1(At_{VA}(a_1)), H_2(At_{VA}(a_2)), \dots, H_D(At_{VA}(a_A)) \quad (4.8)$$

To resolve a query, the resource discovery module composes a multi-dimensional range query, which is the combination of sub-queries on each attribute dimension, and each sub query is responsible to resolve query conditions for a particular attribute in a certain DHT. To perform a range query for a single attribute a_i in the range $[VA(a_i)_{min}, VA(a_i)_{max}]$, resource discovery performs DHT lookup in the range $[H_i(VA(a_i)_{min}), H_i(VA(a_i)_{max})]$. Finally, the discovery results have to satisfy all the single attribute-queries on each attribute dimension. They would be the intersection set between all the individual sub queries results.

Approach C: We just select one attribute dimension and establish a DHT for the appointed attributed dimension. The information of the other attribute dimensions is not necessary to be stored in the DHT, and it will be stored locally at each resource. To perform range query over multi-dimensional attributes, the resource discovery component first performs a range query for the selected single attribute dimension and then the discovery procedure explores the primary set of discovered resources, and chooses the ones that meet all the query conditions for the rest of attribute dimensions. In comparison with the previous approach, this method is better in terms of discovery load and traffic generation because it doesn't generate extra sub-queries. However, the distribution of resource information in the previous approach is more balanced.

Approach D: This approach considers deploying a hierarchical DHT, which consists of A hierarchical lookup layers. The discovery request first explores the top layer of the hierarchy that belongs to a certain attribute a_1 , which has the maximum priority in comparison with other attribute dimensions. It performs a DHT lookup for a single attribute range query in the top layer and then it narrows the exploring space to a certain set of resources that satisfy the query condition for a_1 . In the next level the discovery procedure searches the second level for a_2 , and finally this procedure will end up in last DHT level for a_A . Deploying hierarchical multilevel DHT is similar to the second approach but it creates a significant performance improvement in comparison with the approach B. It eliminates the answers that are not qualified for the single attribute conditions in each level and it makes the exploring space smaller in each level which reduces the discovery traffic load.

Figure 4.22 demonstrates an overall example of the resource discovery message flow for a simple query (i.e., single-resource/single-query), based on the aforementioned communication events. Below is the description of the discovery steps shown in the picture.

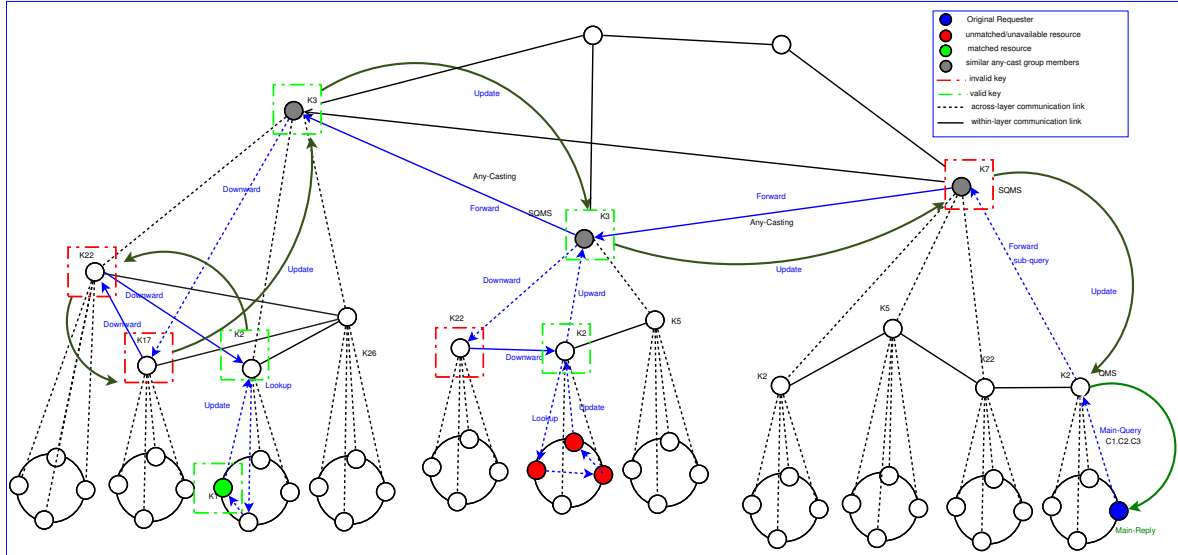


Figure 4.22: An example of sequence of the resource discovery message flow for simple querying (i.e., single-resource/single-query: the main-query contains a single sub-query desiring a single resource with particular query conditions in different layers).

For this example, K_m means a layer-stamp and C_n represents query constraints in layer n . In each layer, the receiver vnode is qualified in accordance with C_n conditions if $m \approx n$ (partial matching) or if $m == n$ (exact matching). Upon receiving a request from a HARD3 user, the RR entity starts the discovery procedure by sending a main-query to its local pre-assigned QMS provider (QMS_{or}). The main-query contains a request for a single resource with particular query conditions in different layers. Accordingly, the QMS_{or} generates a single sub-query and makes a decision for the proper layer to propagate the sub-query. The number of created sub-queries depends on the heterogeneity of the resources requested (i.e., for each group of homogeneous desired resources a sub-query is created). The QMS_{or} decides to send the sub-query to the upper layer by triggering a **Forward** event. The reason is that the sub-query description ($< C_1.C_2.C_3 >$) contains C_3 which is the required conditions for the layer _{sn} and C_3 has the highest priority to be matched compared to C_2 and C_1 (note that, for evaluating query conditions in general, our algorithm gives priority to query conditions with higher layers). In the next step, since K_7 is not qualified, the receiver (which is a SQMS provider) multicasts the query to a group of SNs which layer-stamps (K_3) can be qualified according to C_3 . The multicast member with the shortest latency, receives the **Forward** event and, since C_3 conditions are already obtained, the sub-query is directed to the lower layer by a **Downward** message. The sub-query is traversed among ANs through probability tables until it discovers an AN whose layer-stamp (K_2) can satisfy C_2 conditions. In this case, the QMS starts a DHT lookup in the lower layer by sending a **Lookup** message to a DHT entry. When the lookup fails, an **Update_{qms}** message will be received in the QMS. Consequently, the QMS sends an

Upward message to the higher layer, indicating that the required resource is not available in the lower layers. The sub-query is again anycasted and the discovery continues until all the sub-query conditions have been obtained in all layers (i.e., K_3 , K_2 and K_1). Finally, an `Updatesys` event lets QMS_{or} know about the query results and QMS_{or} consequently sends the discovery results as the main-reply to RR. We elaborate on the details of the various resource discovery algorithms involved in the next section.

4.7 Resource Management

Modern large-scale distributed computing systems are undergoing with the rapid evolution of processor and network architectures. And they have made possible: (i) the integration of more and more cores into one single chip; (ii) many-chips being interconnected into a single machine; (iii) more and more machines getting connected with highly increasing bandwidth. This leads to the emergence of the next generation of many-core enabled large-scale computing systems which rely on thousands of billions of heterogeneous processing cores connected to form a single computing unit.

Current large-scale computing environments such as HPC clusters (e.g., Infiniband-based distributed memory machines, or Bewolf clusters), Grids and Clouds are common scenarios when discussing enhancements to overall computing/system performance and resource/-data/service/application accessibility through efficient sharing and utilization of the integrated infrastructure and hardware resources (such as computing, storage, data and network resources) in large-scale systems, with high heterogeneity (in terms of resources, applications, platforms, users, virtual organization, administration policies, etc.) and high dynamicity. In such large-scale computing environments, resource management is one of the most challenging, and complex issues for efficient resource sharing and utilization, particularly as we move toward Future ManyCore Systems (FMCS).

There are various types of techniques and methods to control and manage the infrastructure for each one of the aforementioned computing environments, which differ based on their main focus, embedded technologies and system architectures. And in fact, designing a resource management architecture which can be applied and adjusted to the requirements of these different computing environments is an extra challenge.

In this section, by “manycore enabled computing systems”, we mean future computing systems that support thousands of chips per compute node and thousands of heterogeneous cores per chip (as predicted in [413]). Referring to the DOS’s concept, discussed in section 4.2.3, we address the problem of resource management for this future large-scale manycore enabled computing environment. Furthermore, the variety of new emergent applications and the heterogeneous resources distributed across the network owned by different organizations and administrative domains, demand that these DOSs support an efficient mechanism for both process and resource management. Designing a DOS resource management system that meets the requirements of such large-scale environments is challenging due to several issues: (a) Supporting adaptability, scalability and extensibility, (b) Incorporating computing resources within different virtual organizations (VOs) while preserving the site autonomy (i.e., the distributed ownership of the resources), (c) Co-allocating resources, (d) Supporting low-cost computation, (e) Supporting Quality of Service (QoS) guarantees.

In this section, based on HARD3, we propose Elastic Core (ElCore), a new elastic, scalable, accurate and dynamic resource management architecture for distributed systems in future

large-scale many-core enabled computing environments, which supports high quality resource mapping and allocation (i.e., resource allocation accuracy) in a fully decentralized manner. In our work, we mainly focus on scalability and accuracy issues, since in our target computing environments (future large-scale manycore systems), one of the most important requirements for resource management is the ability to deal with very large number of heterogeneous resources.

Using ElCore, applications/processes can be distributed among a set of automatically self-organized and self-configured Resource Manager (RM) Entities or Resource Manager Components (RMCs) in the system. RMCs can individually control and monitor their own collection of resources while resources can dynamically be traded (transacted) among RMCs on demand basis. This is achieved by the usage of HARD3, as an embedded dynamic resource discovery method which is able to efficiently discover the most accurate and appropriate resources with the lowest possible latency (i.e., nearest matching resources) among many available resources belonging to different RM entities.

ElCore provides scalability since its decentralized discovery mechanism (i.e. HARD3) allows scalable communications and resource trading between RMC instances. As we discussed in Chapter 3, HARD3 also supports a highly sophisticated query and resource description model (from the very low level of processors and processes to the level of computer clusters with respect to inter-resource and inter-process communication constraints). Using HARD3, ElCore can enable the requesters to accurately discover the resources based on the required specifications and constraints for each given query. These features make ElCore flexible to be efficiently adapted to the requirements of future large-scale many-core enabled systems, be the future HPC clusters, Grids or Clouds.

ElCore also supports elasticity (roughly defined as the ability to address variable run-time changes on resources and workloads) with respect to the following two aspects. First, in our approach, we employ several resource management entities which are distributed across the system in a fully decentralized fashion. Each RMC instance can flexibly control and modify its own pool of resources according to on-demand basis. This gives elasticity to each instance of RMC to dynamically meet various, arbitrary scale, users' resource requirements, ranged from small applications to large computing-intensive tasks (we discuss ElCore policy to trade resources among different instances of RMC in Section 4.7.3). Second, similar to the requirements of the current Cloud computing technology [414], we can expect that one of the most important requirement for application execution in future large-scale manycore enabled systems would be the ability of the system to deal with dynamic applications which have high variability fluctuations in their workloads. As an example of this type of applications we can mention malleable applications [415] where malleability is defined as the capability of the application to dynamically modify its data size and also number of computational entities (threads). The dynamicity of applications (in term of dynamic workload) affects the performance of resource management systems, since resource mapping is based on initial QoS requirements of applications, when workload changes (during run-time), the application execution might fails to meet its initial QoS requirements. ElCore provides elasticity to dynamically detect overloaded resources and reallocate additional resources, allowing migration of processes (like new threads) from overloaded to newly assigned resources (see Section 4.7.5). Furthermore, the modularity and Service-Oriented Architecture (SOA-based) design of ElCore allow it to be coherently integrated and used within DOSs.

4.7.1 Requirements

This section provides some key aspects required to design an efficient resource management model with respect to the requirements of future large-scale manycore systems. It also must be taken into account that our proposed approach does not address all the shortcomings of prior works. Nevertheless, we address the following key aspects for future DOSs:

System Size: In the future, billions of devices may be connected to form a single computing system. This means that, in order to still be able to efficiently use such computing environments, we need to provide scalable approaches which can deal with so very large-scale systems. Thus, scalability becomes one of the most important requirements for resource management for future computing systems [416, 417]. Along this line, decentralized models are generally the approaches that lead to maximum scalability [418], since centralized models commonly suffer from several inherent drawbacks (such as single point of failure and communication bottlenecks). On the other hand, implementing purely decentralized resource management models might hugely increase the level of system complexity (and in some cases it may not be feasible). Distributed hierarchies (which used for our work) can be considered as a realistic alternative decentralized model for resource management, since it can provide scalability while avoiding to overly increase the system complexity.

Heterogeneity: For future massively parallel, distributed and heterogeneous systems, capability of resource management approaches to efficiently handle high heterogeneity of resources is an essential requirement [419]. High heterogeneity of resources (i.e., core diversity) results in increasing the complexity of organizing and managing the resources. For example, different resources, positioned in the same chip or node, may provide different types and levels of capabilities which makes it difficult to apply a common resource management policy to efficiently explore all those heterogeneous resources. In order to solve this problem, we introduce the concept of virtual node (vnode) where each group of homogeneous resources, positioned in a common vicinity, participates to form a single vnode based on given vnode clustering parameters (we discuss this further in Section 4.7.5). In addition, we must note that using a shared memory programming or communication model is not feasible for heterogeneous cores, as well as affecting the scalability of the system. Thus, we assume message passing for communication between cores and chips.

Mapping: For future scenarios, with very large numbers of diverse applications and resources, high quality process-resource mapping can be identified as one of the most urgent problems to be solved [420–422]. The quality of mapping is directly relevant to the accuracy of both resource identification (i.e., precise identification of required resources for each given process) and resource allocation (precise process-resource mapping to meet process requirements). In our work, assuming the former can be dynamically performed by process managers (we discuss this further in Section 5.5), we focus on the later. In order to achieve high resource allocation accuracy, we provide a matching strategy which uses a decentralized strategy (based on distributed hierarchies) to find and map resources. Our mapping strategy supports querying for both design-time (static workload) and run-time (dynamic workload) application requirements through enabling dynamic resource discovery and facilitating process migrations (for overloaded resources). We further discuss our matching strategy in Section 4.7.5.

Elasticity: The elasticity can be defined as the capability of the resource management system to flexibly and sensitively behave on demand basis where the demand comes from resource manager users (requests for resources). Elasticity is one of the desired requirements

for resource management in current Cloud computing systems [423–427]. It must also be considered as an important resource management requirement for future manycore systems, due to the trend of moving Cloud computing concepts to manycore systems. As we mentioned earlier, ElCore provides elasticity with respect to the amount of resources controlled by each resource management entity and the amount of resources assigned to each process.

4.7.2 ElCore Architecture

We propose a novel generic resource management architecture for DOSs, which is able to efficiently manage the allocation of resources in large-scale computing environments with high rate of dynamicity and heterogeneity of resources. We assume a reliable (no message loss, duplication or corruption) and asynchronous message passing model, for inter-core and inter-chip communication in such environments. Our system covers all levels of resources, in all different levels of clustering.

We organize the distributed resources in the computing environment (containing a large number of interconnected resources in different levels) in a set of distributed hierarchies through leveraging our dynamic self-organization and self-configuration approach [294] (see also Section 4.3). We define the entities of Main-Control (i.e., DOS main-kernel) and Nano-Control (i.e., DOS nano-kernel) which are providing maximal and minimal numbers of capabilities, functionalities and services in the system through running each kernel on multiple cores [4, 5, 428]. These control entities differ in terms of service types, which they can dynamically instantiate on demand. The instances of these entities are positioned in the system in a way to map the structure of the underlying distributed hierarchies (i.e., deploying the main-control instances in the hierarchies head-nodes and the nano-control instances in the leaf-nodes).

Similar to the Multitenant Clouds architecture, each (main/nano) control instance provides a dedicated share of the instance including a set of dynamic on demanded services to client machines/components in their region in the hierarchy. In fact, this approach could potentially enhance the multitenancy model for resource management to work in the hierarchies and support dynamic service instantiation (although we do not highlight this aspect since it is out of scope for this dissertation). Moreover, we classify control services as atomic-services (e.g., for the resource management service where every island of distributed resources in the system is managed by a single instance) and non-atomic services. For the non-atomic-services, for instance, we can consider the scheduling service: a local scheduler instance per tile / processor may be responsible for scheduling all code segments within that domain, but will itself be subject to a higher-level scheduler which triggers the respective code block as a whole. The atomic-services are the ones that should not be subdivided in terms of functionalities and workloads.

Figure 4.23 highlights the key services associated to control decisions for resource management (aspects such as deployment instantiation on process movement are important but are not involved in the resource management decisions). These services include:

RR: or Resource Requester is a client-side discovery component, which is responsible for generating, starting and disseminating a query in the system finally returning information about the requested resources (e.g., available qualified processing cores and their respective capabilities). It is an atomic service that can be instantiated by any control entities. Each potential resource client in the system can also have a single RR instance.

RP: or Resource (Information) Provider is a server-side discovery component, which directly offers information services to Resource Requesters and communicates with other

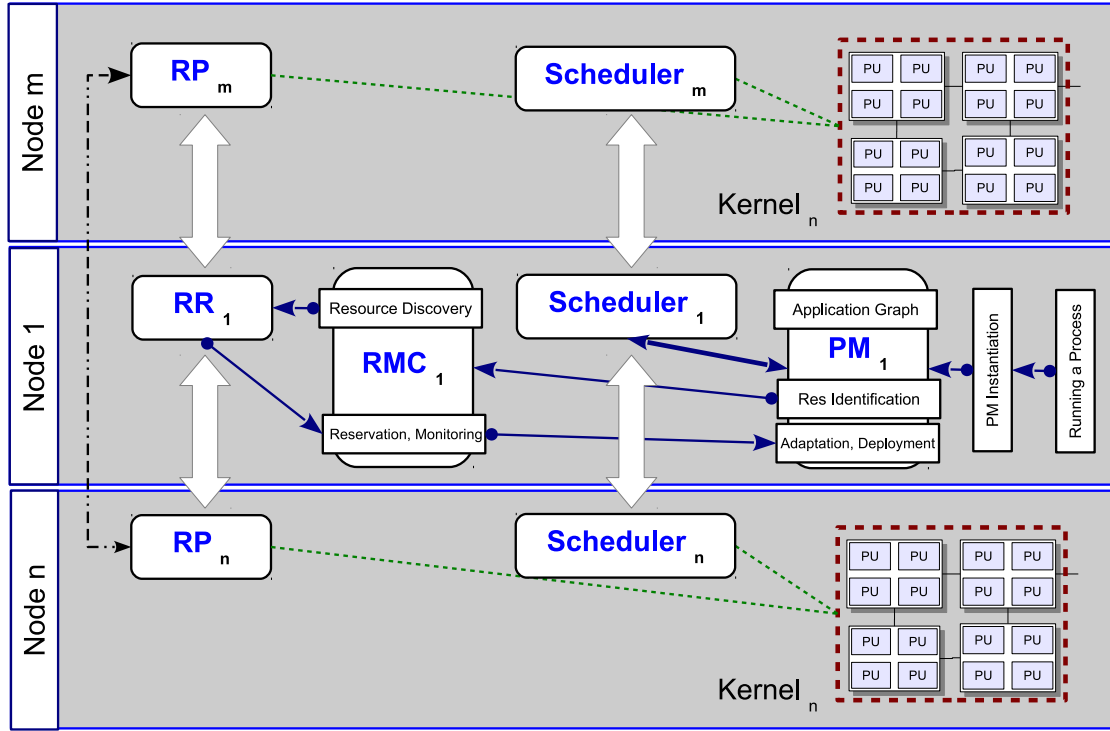


Figure 4.23: ElCore resource management architecture

Resource Providers in the distributed system. RPs collaborate with each other to resolve the received queries from RRs in a completely decentralized and distributed fashion. It is an atomic service which can only be instantiated by main-control instances. However, depending on the specific level of hierarchy, it can be configured as a non-atomic service considering a wider searching space. The RR-RP model is based on self-organizing processing resources in the system, where the computing resources are organized into distributed hierarchies according to a hierarchical resource description (i.e., multi-layered resource description). Moreover, in each layer different algorithms and adapted mechanisms (such as distributed hash tables, distributed probability tables and any-casting) are implemented in order to redirect the discovery requests to the proper requested resources within the layers (e.g., leaf-node, aggregate-node and super-node layers) in the system. More details of our RR-RP resource discovery model have been presented in Section 4.5 [294, 429].

Scheduler is a non-atomic service (since the scheduling loads can be splitted among the other scheduling entities), which is in charge of executing the threads according to a certain order. In general, both types of control entities instances are able to instantiate scheduler instances on demand (e.g., in cluster computing environment, there will be one such component per each processor/resource where the application threads are loaded and needed to be executed.). The details of the scheduling mechanism used are presented in [430].

Process Manager (PM): or Process Manager is in charge of running an application. One such component is loaded and starts every-time the user issues a command for starting an application (i.e., one PM per user per application). The PM is an atomic-service and there would be one single instance for each application. This facilitates the usage of our

architecture in massive parallelized systems, providing a fully distributed application-oriented approach with higher level of customization and adaptation to the heterogeneous types of processes/applications. Using other sibling modules/services, the PM supervises the task of loading an application, interpreting its concurrency structure, allocating the resources, loading the application code on the different resources, computing the scheduling parameters, and launching the application code. Finally, the PM waits for it to finish before de-allocating resources and completing the application execution.

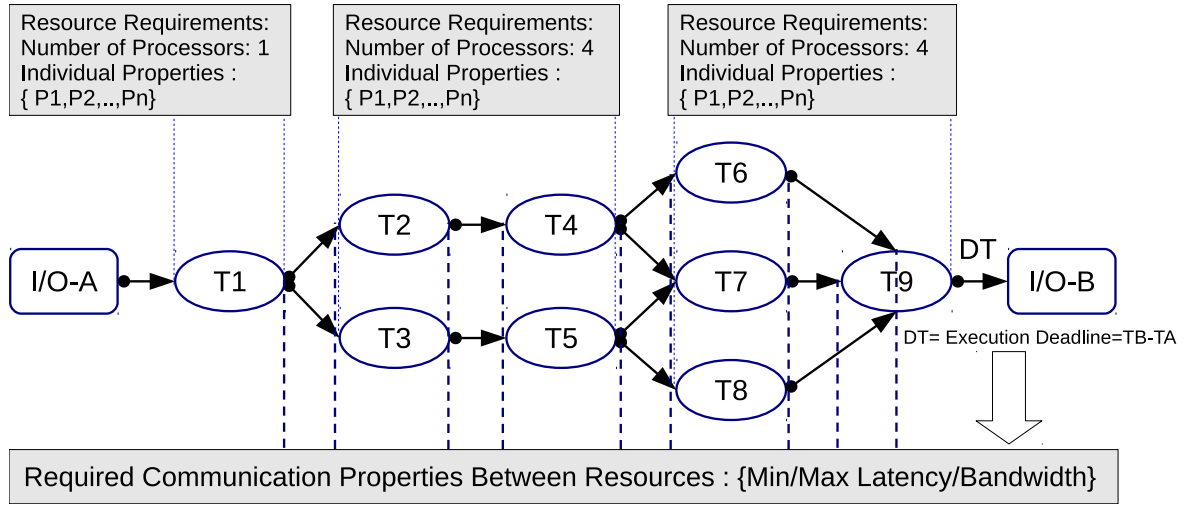
RMC: or Resource Manager Component is an atomic service which can only be instantiated by the main-control entities. Each RMC instance provides a global view of the available resources (i.e., existing capabilities) in the system for its clients while it offers accessibility mechanisms to access those resources and capabilities.

4.7.3 Resource Management Component

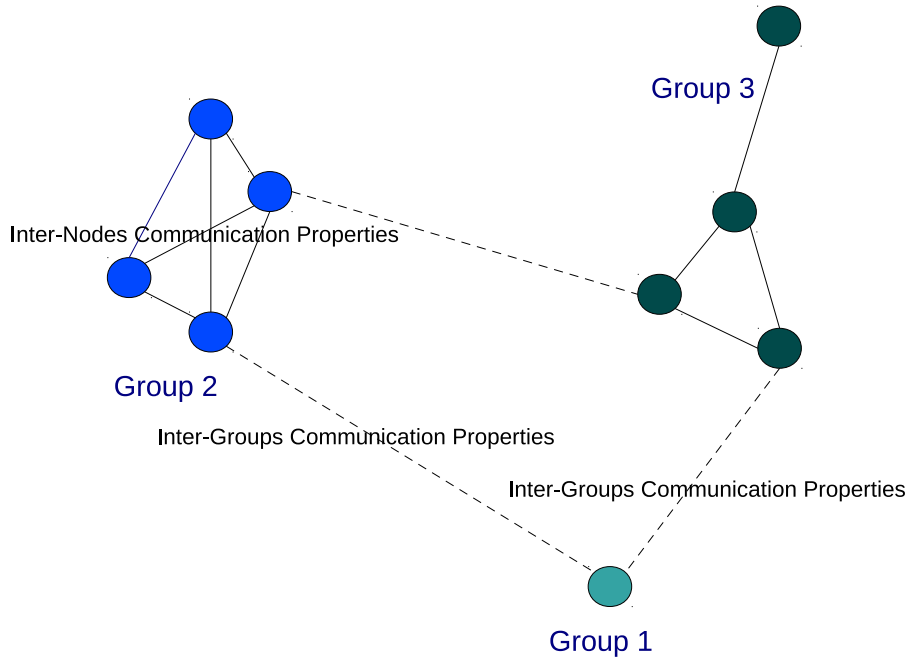
Following this section, we discuss the RMC mechanisms in more detail. RMC manages, monitors and controls the resources in the distributed system. It keeps, monitors and modifies the status of the resources, which are used by the local process managers. It invokes resource discovery components (i.e., RRs and RPs) to find the list of required resources according to the incoming request (wishlist) from Process Managers (PMs) or any other DOS components. It is also able to reserve a single resource or set of resources for a specific process/thread/threads, in order for the result of resource discovery (originated by other remote instances of RRs in the distributed system) not to contain the occupied resources. The reserved resources would be released upon receiving the notification (i.e., notification of process execution completion) from PMs. Additionally, RMC provides a set of capabilities to manage and monitor resources for the other DOS components like the Process Manager. In order to allocate the proper available resources to the application the Process Manager needs to query the Resource Manager.

RMC provides support for distributed management and control of resources. Every pool of resources is managed by one RMC instance, and each instance can provide a global view of all potential resources in the network. It receives requests from many PM components, while each request will specify a set of resource requirements (see Figure 4.24). For example a PM may ask for a minimum and maximum number of processors per each kind, for a certain fraction of the network bandwidth. RMC checks if the required amount of resources is available, and in case of a positive answer it returns a set of resource descriptors that the PM caller module can use to run its tasks. The interaction between the PM and the RMC can be very simple (one simple request that can be accepted or rejected) or very complex (based on a negotiation protocol), depending on the specific DOS configuration.

Algorithm 10 illustrates the RMC resource requesting mechanism when it is invoked by a PM. Upon receiving a PM request, RMC tries to conduct all the requirements of the given request using all the resources it has under control. When it fails to handle a request using the available resources in its own administrative domain (i.e., the collection of distributed resources, which are managed by a RMC instance) RMC starts to extend its global view of resources by invoking a RR component (see Figures 4.25 and 4.26). Consequently RR negotiates with other remote RP instances in order to get information about the required resources which are in the free-ownership-list (i.e., a list of free resources managed by an RMC instance, whose ownership is allowed to be transferred on request) of other RMC instances in the system. If those resources can be found in the free-ownership-list of other remote RMC domains, the local RMC would be able to add them to its own administrative domain.



(a)



(b)

Figure 4.24: a) Extraction of the resource requirements from a given application graph, b) The resource graph of the qualified resources for a given query.

We must note that the resources, which are controlled by a single instance of RMC, are not necessarily positioned in a local node. In fact there, each RMC instance has three main resource collections, which include the reserved-list (or allocated resources), the free-list and the free-ownership-list. The reserved-list maintains the status of the resources allocated to processes. The free-list keeps the information of the resources available for upcoming processes. However, the other remote RMC instances do not have access to these resources. After releasing a resource, it will be moved from the reserved-list to the free-list. Generally, the

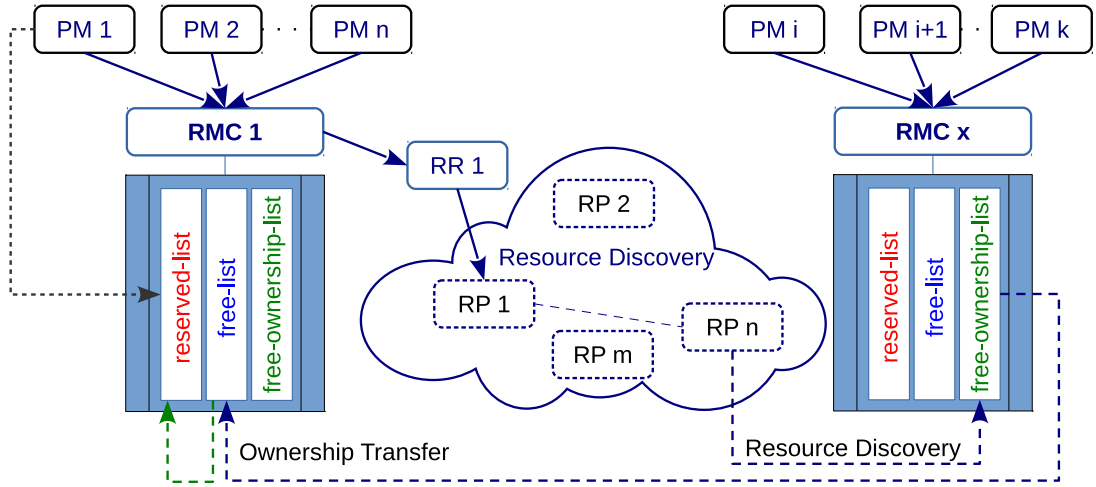


Figure 4.25: RMC mechanism.

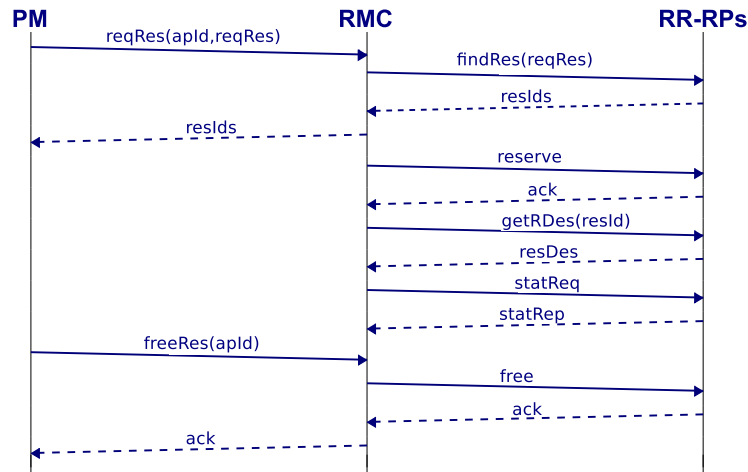


Figure 4.26: Message sequences for the interactions between RR, RP and RMC.

resources in the free-list will be moved to the free-ownership-list after a certain period of time. Nevertheless, depending on the type of processes and their QoS requirements, the RMC can decide to keep some particular resources in the free-list for a longer time. This could be an important feature particularly for deadline-sensitive applications. Examples for this situation would be real-time applications (such as online gaming, voice over IP), where processes must guarantee response within specified timeframes (i.e., deadlines). This means that for such processes, correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed [431] (i.e., a process may fail if has not met its specific deadline). RMC policy flexibility may allow longer maintenance of resources, released by deadline-sensitive processes, in the free-list, for two reasons: first, RMC potentially gets similar requests from PMs for upcoming processes, so that, RMC would be able to respond immediately to those requests without a need to perform a global resource discovery. Second, by keeping a resource in the free-list, instead of free-ownership-list, the scope for resource

Algorithm 10: Pseudo code of mechanism for resource requesting in RMC when PM asks RMC to provide the resource descriptors for a set of required resources.

```

Input: ProcId app, ResourceRequestList rl ; /* process identifier and the description of the required
resources */
Output: ResIDs ; /* res-IDs for the matched qualified available resources */
ResID-Collection reserved-resources; ResId i;
ResID-Collection free-remote-resources; Resource res;
ResourceRequestList rr = rl;
ResourceRequestList request-remote-resources;
ResourceType rt ;
foreach  $rt \in rr$  do
    while  $rt.requested-number > 0$  do
         $i = \text{find-free-resource}(rt.descriptor)$  ; /* searching in the local free-list and
free-ownership-list */
        if  $i == null$  then
            ; /* if the resources cant be found in the local set, the RMC will ask RR to find them
in the system */
            request-remote-resources.add(rt);
            break ;
        else
            dep=analyze-dep-constraints(i,app,rl);
            if dep is not achieved then
                request-remote-resources.add(rt);
                break;
            else
                make-busy(app, i);
                reserved-resources.add(i);
                 $rt.requested-number - -$ ;
    free-remote-resources=RR.FindResources(request-remote-resources);
    foreach  $res \in \text{free-remote-resources}$  do
        free-resources.add(res);
        make-busy(app, res);
        reserved-resources.add(res);
    return reserved-resources;

```

contention for the upcoming similar, deadline-sensitive, requests would be limited only to local RMC processes, and not to all processes around the system. This increases the probability of getting immediate response for those requesters.

The free-ownership-list contains the resources that the RMC will share with other remote RMC instances. Upon receiving a request for a resource, presented in the free-ownership-list, RMC transfers the authority to the requester (i.e., the remote RMC instance) to manage that resource. In fact, despite the free-list, the free-ownership-list contains the only available (free) resources which are allowed to be accessed globally in the system and their ownership can be transferred to the requesters on demand.

4.7.4 Resource Management System Operation

In the rest of this section we elaborate on the inter-operations between the modules/services (which have been previously described in this section) in a general DOS environment and specifically for the discussion under the terminology for the project Service Oriented Operating System (S[o]OS) [4]. Figure 4.27 demonstrates general steps in configuration/inter-operation (i.e., in terms of modules, resources or threads), which contain all the major activities that must be performed for resource management and allocation (i.e., before the actual execution of the code's processes). These configuration/inter-operation steps include the following: a) Analysis of the program and extracting the application's graph. b) Identifying the resources

required for a given process (i.e., identifying the resources, which maximize the matching between resource capabilities and process characteristics). c) Resource discovery, reservation, monitoring and management to find and reserve a matching graph of resources for a given application graph. d) Loading, configuring and instantiating all the necessary modules. e) Deploying the application segments in the reserved resources. f) Distributing code segments and their co-related reserved resources between the assigned sub-schedulers and starting to schedule the allocated threads on the reserved resources.

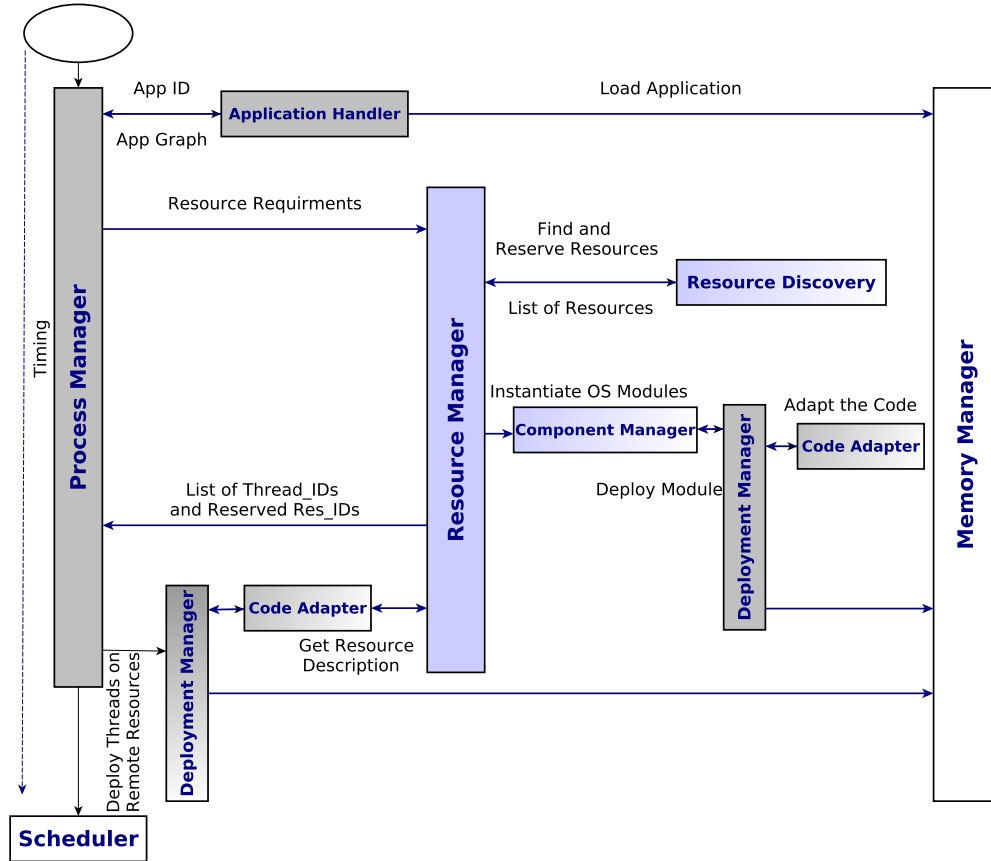


Figure 4.27: Interaction diagram among modules.

As depicted in Figure 4.27, the PM employs the *application handler* to load the requested application. The *application handler* invokes the *memory manager* (or *stream manager*) in order to load the application in memory and then it returns the application graph to the PM. In fact the code is analyzed with regard to its main dependencies, and the resulting application graph can be created with respect to the specific hardware features that would suit the execution best. Due to the application graph, the PM subsequently creates the list of resource requirements, which consists of all the individual characteristics of the potential resource candidate for each vertex combined with the aggregated characteristics and the required communication behavior of all the edges in the application graph. The PM will also pass the minimum and maximum resource requirements (i.e., the individual characteristics of the required resource in terms of processor architecture, type of processor, speed, etc.) for each thread, as well as the minimum and maximum communication properties among threads

in terms of latency and bandwidth (i.e., the inter threads communication requirements) to the RMC. This means that RMC, regardless of the specific requirements of different applications, is able to discover and assign the required set of resources within given ranges (in terms of different computation or communication properties/attributes). In other words, RMC can be queried for resources satisfying arbitrary range conditions on different attributes since it benefits from a range-query enabled resource discovery mechanism. For example a PM may send a query to RMC with conditions such as the following: the required number of resources is 5, core clock rate for each resource must be greater than 1.5 Ghz, L1 cache size for each resource must be in the range of [128Kb, 1Mb] and also maximum communication latency between each pair of resources must be less than 10ms.

In current Grid and Cloud computing technologies, it is very common to specify the minimum and maximum computation requirements (i.e., desired computing attributes) for the resources requested by each query. Similarly, for many real applications, it is very important that the resource management and querying system can perform process-resource mapping with respect to minimum and maximum inter-resource/inter-process communication requirements. For example, for real-time applications, in order to meet the deadlines, we need to ensure that the maximum communication latency among the allocated resources does not exceed a given threshold.

The RMC specifies the resource graph and then it triggers the *resource discovery* module. The *resource discovery* starts to locate resources based on the required computing and communicating conditions. It analyzes all the possible resources and, in order to fulfill all the requirements of the input query, it chooses the most efficient and appropriate graph of resources, by considering the behavior of the communication routes between resources, which follows the communication dependencies (i.e., data source and sink) as identified in the application graph during code analysis. We must note that during the discovery procedure, every discovered resource (within the free-ownership-list of a remote RMC) will initially be marked as “reserved” for a thread in the application graph. This is the process which happens before transferring the ownership of the discovered resources to the local RMC (requester) and it results in a more immediate prevention of other remote instances of the *discovery* module to find the resources that are locally marked as “reserved” (i.e., the discovered resources for a query will immediately be hidden from other queries in the system and if not used in a given time window will be again released). It also must be taken into account that, in our approach, the discovery results (set of discovered resources for a request/process) are deterministic. This means that RMC does not need to either reevaluate the mapping conditions for the discovered resources or select the best matches among a set of candidates (discovery results), rather it simply assigns the discovered resources for the corresponding process.

Figure 4.28 shows an example of resource contention between two RMC instances which require the same resource and initiated at the same time. As we can see in the figure, the resource discovery requests from both RMC1 and RMC2 suddenly and simultaneously arrive at the same destination resource provider (RPn). RPn orders the incoming requests in its own FIFO queue based on the time of arrival (requests that arrive at the same time slot are organized in a generally random order). RPn processes requests one by one. Accordingly, the request from RMC2 will be successful in discovering its desired resource (resource Rx). The state of resource Rx will immediately be changed to “reserved”, thus the next request in the queue (from RMC1) fails to discover the resource Rx. Afterwards, the ownership of Rx from RMCn will be transferred to RMC2.

In fact, once the destination resources for application threads are known, they will be

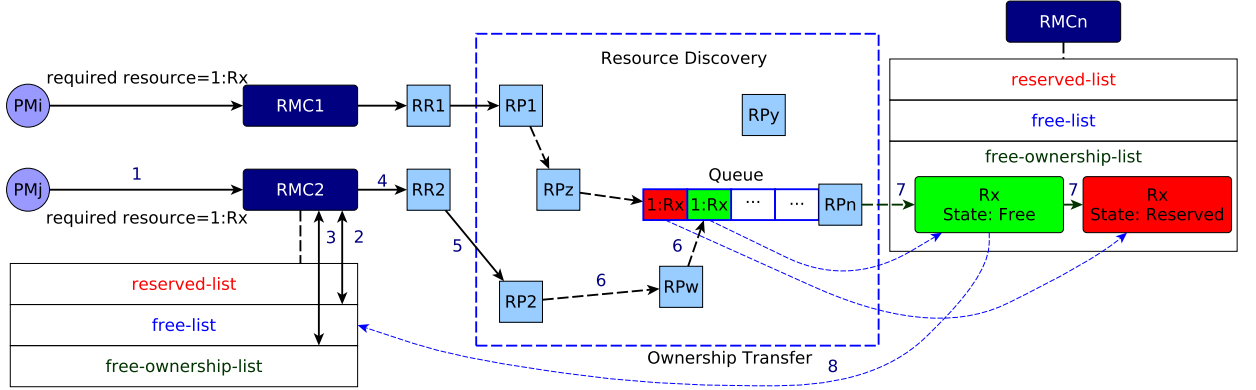


Figure 4.28: Example of resource contention occurs when two (or multiple) different RMC instances (like RMC1 and RMC2 on behalf of PMi and PMj) simultaneously attempt to obtain the same or overlapping set of rare resources (like Rx).

reserved and prepared. All necessary DOS modules for maintaining communication, loading code, executing rendering, etc., will be loaded along with the actual code. For doing this, the RMC will trigger the *component manager* in order to instantiate and setup all the necessary DOS modules. The *component manager* consequently employs the *deployment manager* to deploy the respective modules. The *deployment manager* has the resource description of the target hardware platform (remote resource) and it can invoke the *code adapter* to translate the code and finally write the respective adapted code to the target resource memory, by invoking the *memory manager*. Then the PM triggers the *deployment manager* in order to deploy each of the threads instructions in the allocated remote resources. In turn, the *deployment manager* calls the local RMC to get the full resource description for each resource, and afterwards it starts deploying each thread on the relevant remote resource platform. The PM will finally pass a set of threads on a set of reserved resources to the *scheduler component* in order to begin the actual execution.

4.7.5 Resource Management for FMCS

We can envision that, in the future, very large dimension manycore systems will be constructed by connecting a large number of computing nodes, with many chips and several thousands of heterogeneous cores per chip, using very high-speed network and interconnect technologies. In such FMCS, mapping applications to cores, and adapting each application to the allocated cores plays a key role for an efficient utilization of the computation resources. Efficient resource mapping and allocation in systems with thousands of cores integrated per chip spans a large solution space, and leads to the problem of optimal mapping of parallel applications to the cores which is known to be NP-complete [432].

4.7.5.1 RM Strategy

Our proposed resource management scheme can be leveraged for FMCS to provide a highly decentralized, distributed, scalable and online application mapping with highly resource allocation accuracy. The system is initially configured as follows:

A) In the first step, a virtual hierarchical overlay is created on top of the manycore system by dynamic self-organization of the vnodes (virtual nodes) over the system. Each vnode is defined as a group of homogeneous resources (i.e., processing cores) which are not necessarily positioned on a common physical chip (e.g., a CPU) rather they are positioned within a common vicinity that is described by parameters such as number of hops or interconnect latency (i.e., resources are grouped within virtual-nodes by proximity and similarity). We must note that the resources within a vnode can be also bound to a single common chip. Furthermore, the number of resources within a vnode is bound by a pre-configured threshold value. Therefore, the vnodes with similar resources (i.e., sibling-nodes) might coexist within a single common chip. All vnodes in the system are connected to each other through an overlay topology which is a connected graph created based on the underlying network and interconnect topology during the self-organization phase.

We define two main parameters to construct every vnode which are η and λ . η is the maximum number of resources (cores) in the vnode and λ is the maximum distance between each pair of resources. The maximum distance can be also defined in terms of latency or number of hops.

Depending on the values of the aforementioned parameters, the vnode clustering can be obtained through various approaches. It can be simply achieved by specifying each single core ($\eta=1, \lambda=0$), or all the cores of a homogeneous processor (η =number of cores in the processor, λ =maximum latency among cores in the processor, $\lambda < \delta$ where δ is minimum latency for inter-chip communication) as a vnode. Another way is to simply specify every group of homogeneous cores, positioned in different processors (i.e., chip or die) of a network node, as a single vnode (η =total number of cores in the node, λ =maximum latency among cores in the node, $\lambda < \Delta$ where Δ is minimum latency for communication between cores of different nodes across the network). In fact a vnode is limited to include cores inside a single chip when $\lambda < \delta$ and it is extended to multiple chips when $\lambda \geq \delta$. Apart from that, various dynamic approaches can be also leveraged to perform vnode clustering. In the following, we briefly describe a sample approach for dynamic vnode clustering.

In this approach, for each multiprocessor node in the system, a random single resource (core) is triggered to initiate the clustering process. This resource, which is called primary vnode-head, would be responsible to organize the initial vnode for each physical node (multiprocessor node). The primary vnode-head broadcasts a vnode-clustering-request to all the cores in all the processors positioned in the current physical node and it will receive the description of each resource as well as the latency information from the cores. The primary vnode-head chooses a subset from the responding resources with respect to their similarity and vnode clustering parameters (i.e., η and λ). In this way, a candidate resource of c is selected based upon three conditions: firstly the resource description of c must be matched to the primary vnode-head. Secondly the latency between the primary vnode-head and c must be equal or less than λ and lastly by selecting c , the number of members for the primary vnode-head, should not exceed η . Upon establishing the first vnode, the primary vnode-head specifies the next vnode-head by randomly selecting a resource among the list of unsuccessful candidates. The next vnode-head will be triggered by receiving a message from the primary vnode-head, containing the list of unsuccessful candidates. The operation of the next vnode-head is similar to the primary vnode-head, but what makes it different is that instead of broadcasting to all cores, it uses multicasting to send its vnode-clustering-request only to the resources which were unsuccessful to join the previous vnode-head. The overall clustering process will continue until the list of unsuccessful candidates for the current vnode-head becomes empty. Figure

in a multi-layer hierarchy. The depth of the hierarchy (i.e., number of layers in resource description model) and the definition of each layer might range from very high level (e.g., super clusters, clusters) to very low level (e.g., processing core, ALUs) depending on the architecture designing aspects.

The model is conducted by gathering and combining the individual attributes (ranging from more abstract information in the higher layers to more detailed characteristics in the lower layers) in each layer, augmented with information aggregation and summarization techniques (see Section 3.5), in order to create the layer-stamps. In fact, all specifications (i.e., attributes) of each layer, as well as their values are represented by a single layer-stamp. We also use a similar description model to specify the desired resources by each single query as $\langle c|n_{ln}.c|n_{an}.c|n_{sn} \rangle$. This query-scheme identifies the status of the query conditions for each individual layer in the hierarchy. c_{ln} , c_{an} and c_{sn} denote the existence of query constraints for the layer_{ln}, layer_{an} and layer_{sn} accordingly, while n_{ln} , n_{an} and n_{sn} denote the non-existence of any query-conditions on those layers.

A detailed application description (i.e., application performance modeling) is necessary to accurately specify the desired resources, as well as the minimum/maximum required inter-resource communication capacities for the efficient application execution. By using an appropriate application description model, PMs would be able to dynamically query for the right set of cores for their corresponding processes. Since the application description is out of scope for this dissertation, we assume that the PMs are independently able to initially extract the number and characteristics of required resources, as well as the required inter-resource communication capacities for their corresponding processes at run-time and in an explicit manner. For example (see Figure 4.24), a PM for a process containing 9 threads with different resource and inter-thread communication requirements might ask the RMC to allocate 3 different groups of homogeneous cores to fulfil both the inter-group and intra-group constraints for communication between allocated cores. We also assume that applications/processes consist of tasks/threads that are executed on a single core and that may be freely re-allocated to different cores.

4.7.5.3 Matching Strategy

When a RMC receives a resource request from a PM, if the demanded resources are not in the local free-list and free-ownership-list, the RMC invokes a RR component to perform a resource discovery. RR in turn sends the given query to a RP-QMS, where the RR host (i.e., a vnode which hosts the RR service) is a member of its LN group. Due to the type of query and query demand (e.g., simple single resource, multiple heterogeneous resources, complex resource graph, containing constraints for inter resource communications) the RP-QMS splits the main-query in a set of sub-queries, and chooses individually the appropriate layer for each sub-query to start being processed. The query processing in layer_{an} is based on distributed probability tables that facilitate dynamic distributed learning processes, which are adapted to progressive environmental changes. In layer_{ln}, LN members participate in a specification-based DHT ring where the sibling nodes are linearly organized in linked lists, with single entries on the DHT ring, and where vnodes are positioned in DHT based on their core-level specification stamp. The query processing and forwarding in layer_{sn} also leverages the specification of the resources at the node-level, to conduct a specification based anycasting method and direct the queries among SNs.

The matching strategy of the resource discovery mechanism is based on matching the given

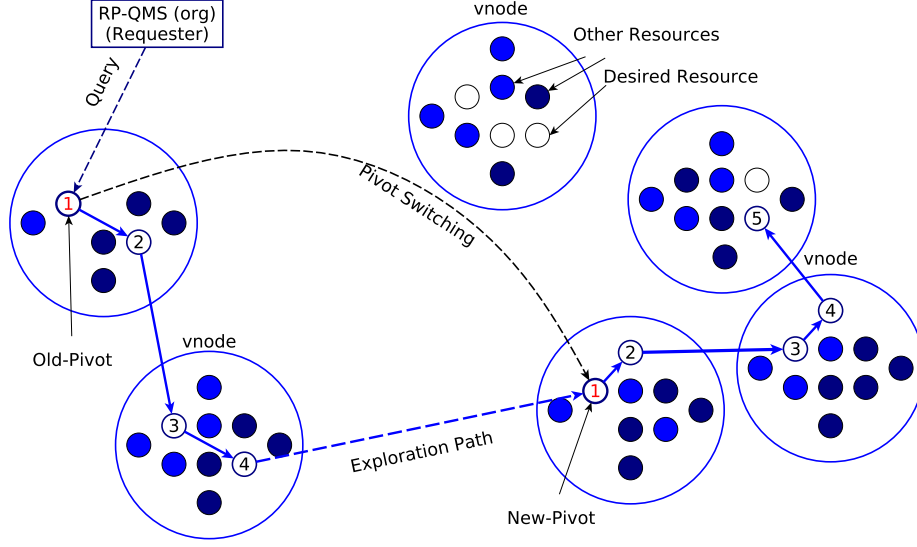


Figure 4.30: Example of matching strategy for inter-resource communication requirements using Pivot Switching mechanism. A sub-query, for 5 homogeneous resources with given maximum inter-resource latency, initiated from $RP-QMS_{org}$ and explores vnodes (from the closest vnodes to the most far one). The sub-query switches the pivot, when the recent matched resource fails to meet the upper bound communication requirement (maximum latency).

query descriptions for each layer (desired attribute conditions for each layer) to their corresponding layer-stamps, at the time of visiting every individual vnode in different hierarchies. As we mentioned earlier, each hierarchy consists of three layers and every vnode depending on its role in the hierarchy (LN, AN or SN) provides layer-stamps. The exploration process for every sub-query (i.e., a sub-request, aiming to discover a set of homogeneous resources within a common vicinity) starts from the nearest, potentially qualified and available, resources to the far resources with respect to giving matching priority from the highest layer to the lowest layer ($layer_{sn}$ to $layer_{ln}$). This way guaranties that the nearest resources (to the origin requester) are discovered in advance. Each sub-query may include a number of required resources with similar specifications as well as inter-resource communication requirements (e.g., maximum communication latency among desired resources). For each sub-query, in order to match the given communication requirements to the inter-resource communication links among the matched resources, we use the following approach:

- 1- Sub-query proceeds to explore vnodes/resources (from nearest resources to far resources), regardless of its communication conditions, focusing on computation requirements (query requirements for different layers).

- 2- Upon discovering the first match, sub-query sets it as the initial pivot (reference) for measuring inter-resource communications.

- 3- Sub-query continues to discover the rest of required resources and when it found the next match, it makes a decision, by first evaluating lower bound, and then evaluating upper bound of inter-resource communication requirements, to perform one of the actions listed: ignore the current matched resource, perform pivot switching or continue the normal search. Sub-query ignores the current matched resource and continues the discovery process, if the

current matched resource fails to meet the lower bound requirement (minimum latency). If this doesn't happen, sub-query must next evaluate the upper bound condition (maximum latency). Accordingly, pivot switching might be required. Pivot switching is the operation for changing the pivot of a sub-query to the current matched resource (see Figure 4.30). By doing a pivot switch, sub-query ignores all previously matched resources and adds the current matched resource, as the first match, to its list of matched resources. Afterwards it proceeds to discover the rest of required resources. Pivot switching happens when the current matched resource fails to meet the upper bound requirement (maximum latency). If pivot switching doesn't happen, the discovery process proceeds normally, until all required resources are detected.

Finally, the RP-QMS that originated the sub-queries, aggregates the discovery results and replies to the RR with a set of resource matches that optimally satisfy the main-query demand. After acquiring the discovery result from the RR, RMC starts to perform a resource trading operation. It begins to negotiate with other related RMC instances in the system which maintain the discovered resources in their free-ownership-list and consequently transforms the ownership of those resources to its local free-list.

Each RMC instance periodically monitors the status of resources in its own resource pool, through a HotSpot-detector component which is responsible for detecting the overloading resources in the resource-pool. Whenever a HotSpot is detected, the RMC re-evaluates the resource requirements for the running application. Accordingly the RMC allocates set of new resources for that application, and starts to migrate processes to the new resources. This mechanism provides elasticity for ElCore to be able to tolerate the application with dynamic behavior (aka malleable applications).

4.8 Conclusion

In this chapter, we proposed HARD, a novel efficient and highly scalable resource-discovery approach, which deals with the resource discovery requirements in large scale computing environments such as high-hierarchy, high-heterogeneity and high-scalability and dynamicity. The approach is based on self-organization and self-adaptation of processing resources in the system, where the computing resources are organized into distributed hierarchies according to the proposed hierarchical resource description model (i.e., multi-layered resource description) in Chapter 3.

In Section 4.2 we elaborated on the HARD requirements and design principles, which contains our primary assumptions and the proposed system architecture. HARD architecture deploys various layer-based hybrid adaptive mechanisms (i.e., inter-layers and intra-layers methods) in order to efficiently direct discovery requests to the proper resources across layers. Due to properties and characteristics of each layer in the hierarchy, HARD presents a set of specific adapted methods which have particularly been designed to obtain the maximum discovery efficiency on the target layer, while an integrated and coherent approach is used to traverse layers in hierarchy.

In Section 4.3 we presented our hierarchical proximity-aware self-deployment and self-configuration algorithm, to alter the underlying network topology in a way that the optimal clustering of the nodes in the different layers can be performed dynamically. The simulation results show that our grouping mechanism supports load balance while it composes the groups in the layers, with respect to the required locality and clustering requirements. Furthermore, the experiment results prove the scalability and efficiency of the proposed algorithm for

different system sizes.

We presented the details of the proposed mechanisms and algorithms for HARD operations in various layers of the hierarchy in Sections 4.5 and 4.6. We presented different algorithms and adapted mechanisms (such as distributed hash tables, distributed probability tables and anycasting) in each layer and with respect to two HARD's main modules (RR and RP) in order to redirect the discovery requests to the proper requested resources across and within layers (e.g., leaf-node, aggregate-node and super-node layers) in the system.

We have also presented a novel resource management architecture for large scale distributed computing environments in Section 4.7. The proposed architecture contains a set of modules, which will dynamically be instantiated on the nodes in the distributed system on demand. Our approach is flexible to allocate the required set of resources for various types of processes/applications. It is a generic resource management approach, which can be applied to different large scale computing architectures and specifically it can be explored in Cloud systems. We must note that we have not specifically designed our approach for Cloud computing only. Instead, the proposed resource management architecture provides a generic solution (considering the general requirements of large scale computing environments), which can bring a set of interesting features (such as auto-scaling, multitenancy, multi-dimensional mapping, etc.,) that facilitate its easy adaptation to any distributed technology (such as Service Oriented Architecture (SOA), Grid and HPC many-core). Our resource management scheme can support auto-scale (the ability to provide budget to more-or-less expensive tasks) in two different aspects: dynamic resource allocation and dynamic module instantiation. Using our approach, the mapping between processes and resources can be done with high level of accuracy which potentially leads to a significant enhancement in the overall system performance. System module instances would be automatically created when they are needed. Each module instance would be dependent on the type of application (e.g., real time, HPC, etc.), the computing resource (where the application starts to be executed), heterogeneity of resources, positioning of the resources in the system, network topology and interconnection between resources. Moreover, leveraging discovery components (RR-RPs) enables our resource management platform to dynamically find and allocate available resources that guarantee the QoS parameters on demand.

We evaluate the performance of HARD and ElCore over highly heterogeneous, hierarchical and dynamic computing environments with respect to several scalability and efficiency aspects in the next chapter (Chapter 5).

Chapter 5

Evaluation

This chapter presents the simulation results for evaluating the proposed resource discovery and management. For this, we use simulation instead of experiment on the real large scale computing infrastructures (e.g., PlanetLab, TACC, Oak Ridge, BSC, GENCI and public cloud providers like Amazon and Google) due to issues as cost and flexibility of the simulation to design, develop and assess several algorithms as well as providing full control over system behavior and evaluation scenarios. On the other hand, real infrastructures generally provide limitations to explore the design space, particularly for scalable performance and large scale evaluation. Due to these reasons, we consider the evaluation of our discovery approach on the real infrastructures as future work.

5.1 Introduction

To do our evaluation, we developed a simulation platform, based on OMNET++/INET-Framework and Oversim simulation tools, which is able to simulate many-core environments (up to 55000 processing cores in different chips and network-nodes), focusing on communication aspects (i.e., communications between cores, chips and nodes), albeit taking long simulation times (a 3-week simulation run is average). We conduct and implement our experiments in different scenarios and settings. Accordingly, we evaluate the scalability and efficiency of HARD3 with respect to the several evaluation criterias (such as discovery latency, discovery load, discovery accuracy and discovery cost) in large scale resource pools (containing 1000 to 55000 processing cores positioned in 100 to 1000 network-node) with presence of high dynamicity and high heterogeneity of resources and show the performance achieved with the proposed methods and heuristics. We will evaluate scalability (for both synchronous and asynchronous querying) and efficiency (for complex querying in high heterogeneous computing environments) and finally we compare our discovery approach with other different proposals. Figure 5.1 illustrates the general steps involved in developing evaluation strategies for different components in our research work.

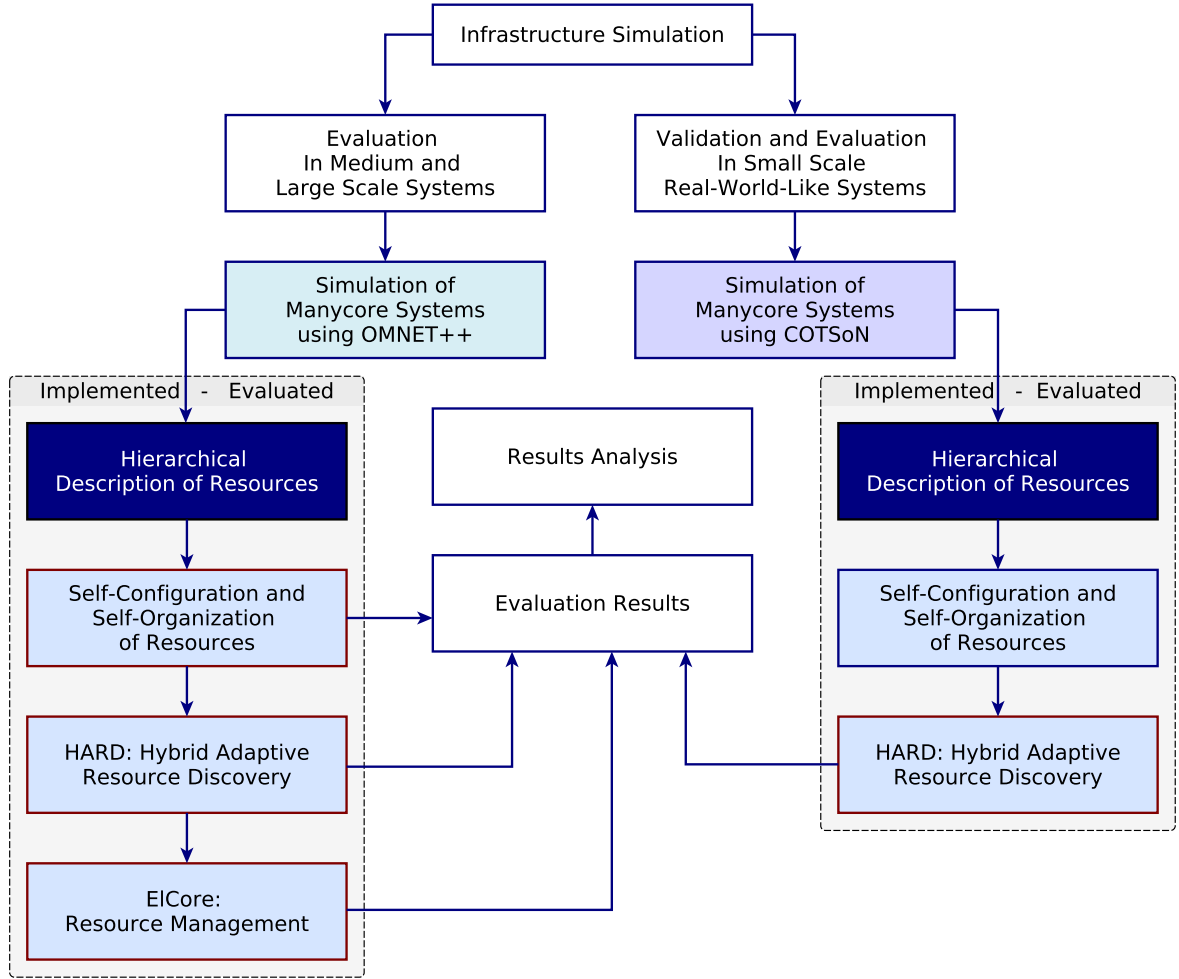


Figure 5.1: Overview of the evaluation strategy.

5.2 Simulation Environments

We use SimNow, COTSon, Omnet++/INET-Framework and Oversim simulation tools to conduct and implement our experiments in different scenarios and settings. The following subsections present brief description of each of the aforementioned simulation tools.

5.2.1 SIMNOW and COTSON

AMD SimNow [9] is a fast cycle, accurate full system emulator, using caching and dynamic compilation techniques. It can support booting an real operating system and launch complex applications over it. The SimNow emulator supports the x86 and x86-64 instruction sets, with support for other devices of a real system. It performs emulation of the real system with (at least) 10x slowdown in comparison with the native execution. SimNow cores generate a series of event that are stored in asynchronous queue.

COTSon [8] is the proposed HP labs' full system simulator based on AMD SimNow.

It allows to trade speed for accuracy depending on the simulation purpose and the user preferences. It uses a trace driven approach where a single core, full system, simulator generates thread instruction streams. This helps COTSon achieving a better simulation speed, compared to execution-driven simulation, because of the decoupling of functional simulation from detailed simulation. It is also possible to simulate any application that might have been run on different platforms, given the correct tracing infrastructure. COTSon provides timing feedback for SimNow instances, it parses asynchronous queues to create higher level objects such as instructions. COTSon is a complete tool providing timing information back to the functional emulator in order to affect the behavior of the application. It also uses Quantum based simulation for synchronization techniques. In this way, every time a quantum starts, the timing module will get a notification about the starting time and the quantum length. Similarly once a node ends a quantum, the timing module will let the other modules to know about the network timing information, which has been calculated during the past quantum. The functional simulation adds extra latency to all the submitted packets, because network packets are sent twice. The source NIC sends packets to the mediator timing module and the mediator will send the packets to the target NIC module. Additional time is required for the packet processing in the mediator, therefore COTSon uses a Quanta (Q) bigger than real latency time between two nodes.

5.2.2 OMNET++

OMNET++ is a component-based discrete-event simulation library and framework for simulating networked systems. It provides developers with the capability to define highly reusable modules, which can be instantiated and interconnected in different designs. Modules have gates for interaction with other modules and can be combined with each other at arbitrary nesting levels. Modules communicate through message passing via defined gates and channels (connection between modules). Modules, channels and gates can be parameterised by leveraging the Network Description (NED) Language, and used for customizing the behavior according to specific application scenarios (see Appendix A for more detail on simulation using OMNET++).

5.3 HARD Evaluation in Small Scale Systems

In this section we describe simulation results regarding the resource discovery evaluation in small and medium scale many-core-enabled computing environments. A key factor in our evaluation is showing the functionality and scalability of the Resource Discovery (RD) system. To achieve this we have set up a virtual networked environment with a number of multi-core/multi-processor nodes, where all the processors are enabled to invoke the resource discovery module, in order to find other possible alternative resources for the transaction of the overloaded processes to remote processors. The simulation experiments have been conducted based on the COTSon Simulator and we simulated a networked environment with 23 multi-core nodes (each node has 2, 4, 6 or 8 cores) including the total number of 88 resources (cores). Scalability for RD means the ability of the discovery mechanism to support a larger number of resources or a greater number of inter-operations between the nodes, or both. We analyze the RD scalability by measuring discovery latency versus the number of nodes where one, or a constant fraction of the nodes, or a percentage of the total nodes, concurrently and periodically (with different intervals) generate a query and trigger the resource discovery

module to get information about other nodes in a congested network. We analyzed the impact of the network size on the discovery delay which is the response time to get information.

To achieve scalability, discovery delay should not have a big variation when we change the size of network. We evaluated the impact of the network size on the resource discovery latency for different number of requesters. After a random initialization time, each requester starts submitting a query to the system every 30 s. We have run 1000 RD queries for each requester in the given network size. The minimum query dimension is 4 and the maximum number of attributes is 20. We also vary the number of desired resources for each resource discovery query from 1 to 4. The properties of each query and also the network topology are random. Accordingly, depending on the network topology and network size in each run, the simulation scenarios with different number of resource information providers (i.e., the nodes with QMS support) as distributed directories for our simulated network, would be established. However, the number of resource providers would be fixed during the simulation run time.

In Figure 5.2, while we increase the number of resources, the discovery response time starts to increase, but for larger number of resources it remains almost constant. In all these experiments, that were done with different number of requesters, we roughly see the same behavior in the results. This shows that discovery latency is independent of the network size, which means that for large scale networks we could still have a reasonable response time for resource discovery queries.

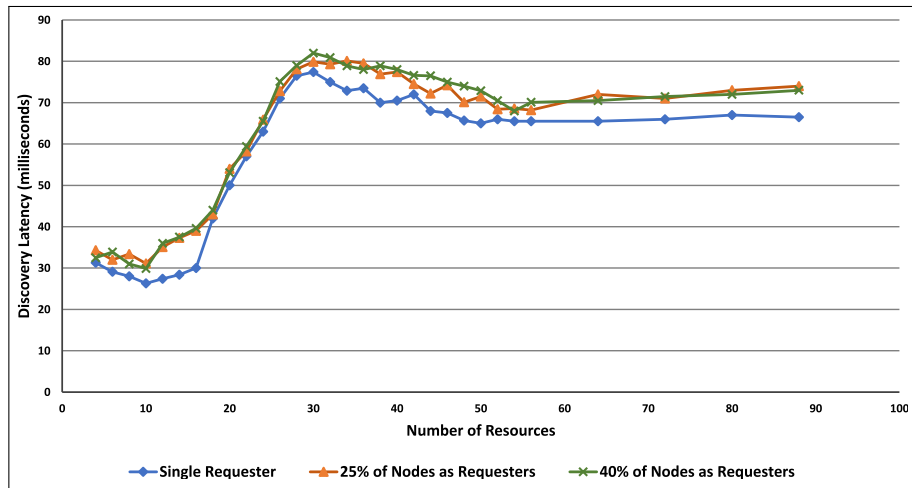


Figure 5.2: Average discovery latency for single requester, 25% and 40% of nodes as requesters with interval 30s.

In the second test we measured the average of the discovery overhead versus the number of nodes. This test aims to analyze the impact of network size on the average number of transaction messages for each discovery process. Figure 5.3 plots the discovery overhead of our system network sizes in the two configurations. In the first configuration only one of the nodes generates a random query every 30s. By increasing the network size first, the overhead (the average number of discovery messages per query) increases until it gets to a threshold and then it decreases slightly and remains almost constant for large network sizes. By increasing the network size the RD system is required to forward the queries to other remote directories with larger distance in the system to find the resources around the network.

Therefore, increasing the network size leads to increasing the search radius, which would eventuate in propagating more messages for resolving the discovery requests by efficiently exploring the discovery region. Thus we could expect to see the growing of the overhead with the increasing number of resources.

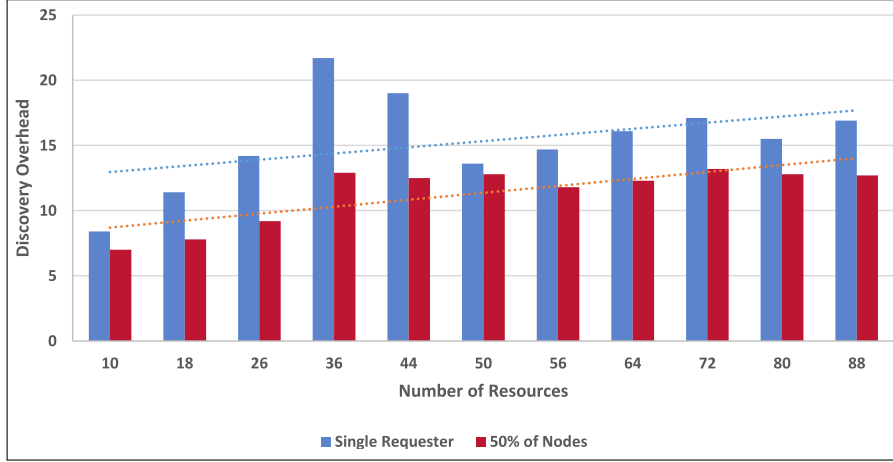


Figure 5.3: Average discovery overhead in different network size for 1 requester and 50 % of nodes as requesters with query rate=.033/s.

The overhead starts to reduce after reaching the maximum, due to the increasing number of remote directories that provide information about other alternative resources, and also the increase of the resource replication. Interestingly, in a second configuration that we have simulated for a congested network with 50% of the nodes as concurrent requesters, the results show a significant decrease in the discovery overhead. The reason for this overhead reduction is that when we perform several queries concurrently, by different distributed RD components (i.e., resource requesters), the probability tables of the intermediate directories (QMS nodes) would dynamically be updated according to each discovered result. This will improve the degree of resource awareness in the probability tables in the directories which would lead to the reduction of the discovery overhead (i.e., reducing the number of forwarding and query dissemination) compared to the first configuration. Moreover, implementation of the resource caching mechanism in nodes of different types (either LN, AN or SN nodes), enables general nodes to store the result of successful queries for a specific period of time, which facilitates the resolving of similar queries from other requesters with less overhead cost. In both configurations the overhead variations are reasonably small for large network sizes. Therefore the overhead should not be dependent on the network size, which means that our RD solution is scalable for small scale systems.

5.4 HARD Evaluation in Medium and Large Scale Systems

5.4.1 Scalability for Synchronous Querying

Upon starting an iteration in a synchronous querying approach, requesters simultaneously start to propagate their discovery requests (i.e., a single main-query per requester per iteration) in the system. In contrast to asynchronous querying, the reserved resources (by requesters) in

synchronous querying would be released after timing synchronization and before starting each successive iteration process. However, the distributed probability information, gathered in the whole system during an iteration will be maintained to be used and updated in the successive iterations. The purpose of our evaluation under these assumptions is to fully understand and precisely study the HARD’s scalability behavior with minimal impact caused by resource reservation.

5.4.1.1 Simulation Setup

We conduct our simulation experiments along two different main scenarios (i.e., synchronous and asynchronous querying scenarios), and for each scenario we conduct several experiments with regard to different simulation parameters.

For evaluating the synchronous querying we simulate a many-core networking environment containing between 10 to 100 types of heterogeneous computing resources which are uniformly distributed within a system, with the size ranging from 1000 to 10000 resources, and where each simulated node represents a computing resource. The simulation starts by performing a self-stabilization of the nodes during the initialization phase, which leads to the establishment of a distributed hierarchy of virtual overlays on top of the simulated network. Upon completion of the initialization phase, nodes in the system would be qualified to start discovery requests.

Table 5.1: Simulation parameters for scalability evaluation (synchronous querying).

Parameter	Values
Maximum start-up time	3000 ms
Querying iterations per requester	4
Querying interval	2000 ms
Complexity of querying	single-resource/single-query
Percentage of requesters	1%
Frequency of Target Resources (FTR)	30,60,90,120
Physical network size	1000 to 10000 cores
Interconnect topology	mesh/torus
Network topology	random
Interconnect channel data-rate	50Gbps
Routing type	DOR
Network channel	100 Mbps

We schedule 4 synchronous querying iterations (time periods) for all the requesters in the system. A constant fraction of nodes (1%) are randomly selected to initiate discovery requests by specifying their resource requirements (in terms of number of target resources to be discovered, arbitrary level of details of computing characterization factors and communication properties among requested resources) as desired target attributes (in different resource description level). The selected requesters, after a synchronization step, regenerate the similar queries in the next 3 iterations. Moreover, the scheduled queries in all iterations for all the requesters are also uniform with the type of single-resource-single-query. The frequency of the target resources are ranged from 30 to 120. The other simulation parameters for the first scenario are summarized in Table 5.1.

5.4.1.2 Simulation Results

In the first experiment we evaluate the impact of changing the system size (in terms of the total number of resources in the system) on the discovery cost, in terms of the number of

required messages to perform a discovery request. As it is shown in Figure 5.4, the discovery cost in all the tests is decreased in the subsequent iterations. In other words, the average number of required messages per discovery request decreases as time progressed which means that the proposed discovery approach is scalable over time. The reason for this is that the query guidance mechanism in the DPTs will be dynamically improved, by enhancing the probability values for successful discovery over larger number of query dimensions, as well as raising the quality of SoR preferences in the QMS providers in the system with respect to the results of both the past queries in the former iterations and the concurrent queries from other requesters in the current iteration.

The discovery cost reduction is specially significant when time progressed from the first iteration to the second iteration, while for the subsequent iterations, the speed of discovery cost reduction is decreased gradually with a lapse of time. This happens because of the lack of querying knowledge in the DPTs for the first iteration. In fact, DPTs should be tuned and warmed up (i.e., build up) through initial queries, before they can efficiently be used in the system. As we can also see in the similar results of a experiment, for measuring discovery latency in various system sizes (see Figure 5.5), the initial DPTs warm-up leads to higher latency for the initial queries which are generated in the first iteration. However, depending on several system parameters, such as discovery traffic (in terms of the number of discovery requests, number of requester, querying intervals, etc.), complexity of the queries, heterogeneity of the resources and DPT's configuration (in terms of predefined resource-types, SoR capabilities, etc), DPTs can quickly be adjusted and updated, in order to predict and recognize the appropriated high quality SoRs in the system for each given discovery request, with higher level of accuracy, and even the DPT warm-up cost might not be visible after several number of querying iterations.

The results in Figure 5.4 also show that the average number of messages required per discovery request, for different system sizes, increases due to the increased exploring space of the larger systems. However, in all tests, for larger iteration numbers, and specially for the larger systems, the slope (in most places) is very slight or steady, which demonstrates that the proposed discovery approach can tolerate an increase of the system size (in terms of number of resources), while it maintains system efficiency by exploring the larger search space for the discovery requests, with almost constant number of messages. Moreover, the average number of required discovery messages decreases when the Frequency of the Target Resources (FTR) is getting higher. Considering the similar behavior for the latency tests in Figure 5.4, it can be seen that HARD3 provides good scalability for the discovery requests in large and very active systems.

Furthermore, the results presented in Figs 5.4 and 5.5 prove that our proposed DPT mechanism is fault tolerant. As depicted in these figures, a large change (improvement) in the discovery performance in the second iteration has happened compared to the discovery performance in the first iteration. This improvements continues in the next iterations. But, as it shown in the results, after a few initial iterations, changes become imperceptible. Indeed, DPTs are empty at the beginning, but after a few iterations, they can rapidly become informative and provide a reasonable stable performance. In other words, altering probability tables in some nodes does not have a visible impact on the overall system behavior. This makes probability tables more powerful for being rapidly recovered from any potential failure.

Figs 5.6-a and 5.6-b demonstrate the overall results of all aforementioned iterations including the DPTs warm-up costs to measure the impact of system size on the discovery overhead and discovery latency accordingly. As we can see in these results, the mean discovery

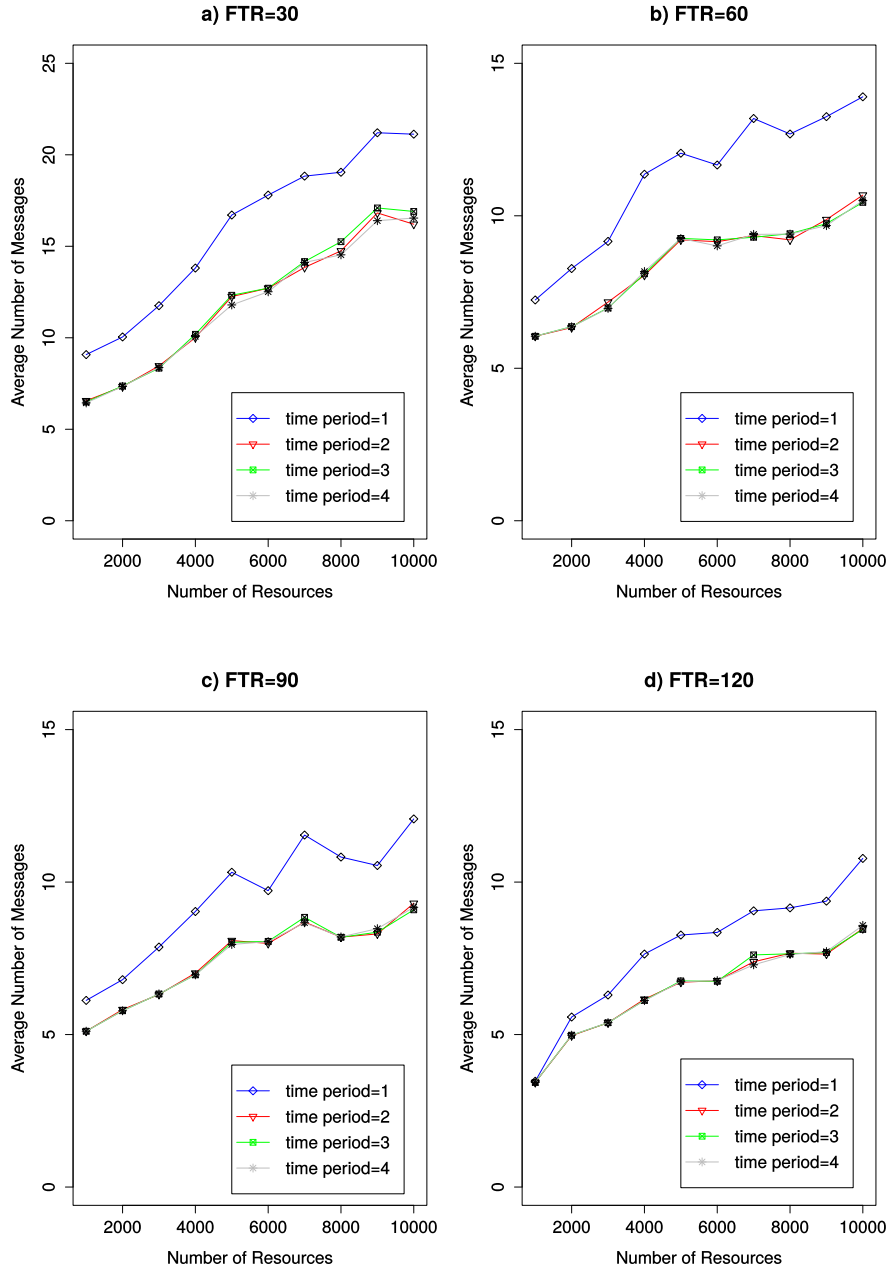


Figure 5.4: Average number of discovery messages vs system size for basic querying (i.e., single-resource/single-sub-query) in several querying iterations (time periods) with different FTR values. FTR is the overall frequency of the target (desired) resources.

overhead and the mean discovery latency, particularly for the reasonable frequency of the target resources (like $FTR > 60$), and for the larger system sizes, remains stable with no significant changes.

The overall results in this section proves the HARD3 scalability at least for the simple querying. In the next section we evaluate the HARD3 scalability under complex querying

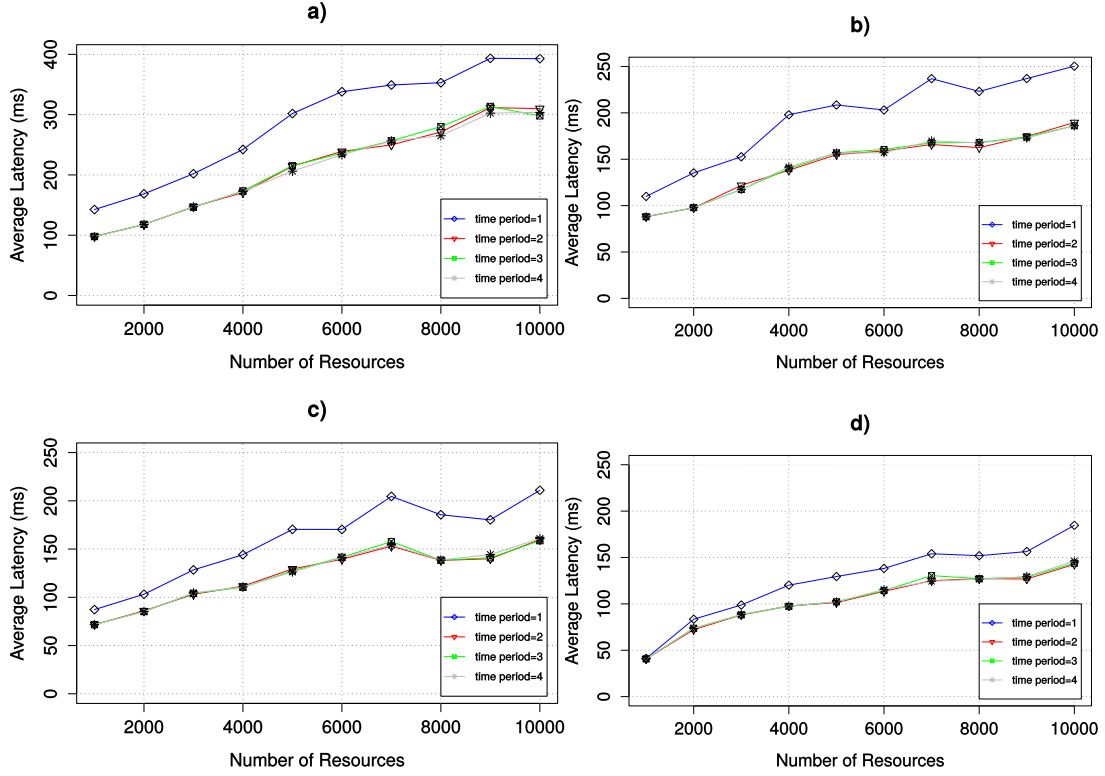


Figure 5.5: Average discovery latency vs system size for basic querying (i.e., single-resource/single-sub-query) in several querying iterations (time periods) with different FTR values. FTR is the overall frequency of the target (desired) resources. a) FTR=30, b) FTR=60, c) FTR=90, d) FTR=120.

conditions whereas high resource heterogeneity is emphasized.

5.4.2 Heterogeneity and Complexity

HARD3 provides flexibility for complex querying in terms of multidimensional querying, resource graph querying, exact/partial and range querying. Multidimensional and resource graph querying are inherent features of HARD3 due to the original attribute-based definition of resources in HARD's hierarchical architecture and mechanisms as well as the query description. In other hand, in addition to exact querying, the partial and range querying are possible by leveraging a similarity function in different layers, where HARD3 algorithms would be able to discover the required resources with a certain amount of approximation.

Indeed, requesters are allowed to send main-queries to their corresponded QMS providers, containing multiple querying conditions in each layer, as well as inter-resource and inter-resource-group communication constraints for the desired resources. A main-query might includes the required conditions for several different (i.e heterogeneous) resource groups where each resource group specifies the number of required resources with similar (i.e., homogeneous) characteristics and inter-resource communication properties. QMS providers, in turn, split the main-queries into various number of sub-queries depending on the number of heterogeneous resource groups described in the main-queries. The obtained concurrent

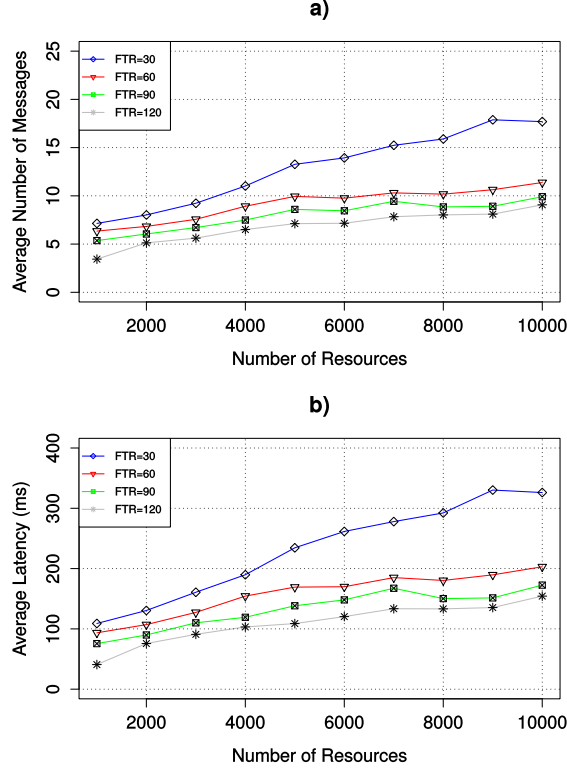


Figure 5.6: a) Discovery overhead (in terms of number of required discovery messages) vs system size for different frequencies of the target resources in synchronous querying, b) Discovery latency vs system size for different frequencies of the target resources in synchronous querying.

sub-queries dynamically and independently choose their own-path across the system in order to find their required number of homogeneous resources and finally returns the discovery results to their origin QMS provider, which is responsible to maintain the overall-state of the main-query and make proper decisions accordingly. In this section, we evaluate the impact of complex querying on the HARD’s scalability by increasing the number of desired heterogeneous resource-groups, as well as the number of required resources per each group in the main-queries of the requesters. We provide evaluations for both uniform and non-uniform distribution of SoRs capabilities (i.e., initial number of potential available resources in each source of resource). In uniform distribution, SoRs in the system initially have equal capability to provide resources to the requesters, while in non-uniform distribution the initial capabilities for each SoR might be different.

5.4.2.1 Simulation Setup

In order to evaluate the HARD3 performance with respect to high complex querying and high resource heterogeneity, we conduct a simulation scenario based on the previous scenario, but with some added changes. Simulation parameters presented in Table 5.2.

Table 5.2: Simulation parameters for HARD3 evaluation under the complex querying conditions and high heterogeneity of resources (synchronous querying).

Parameter	Values
Complexity of querying	multi-resource/multi-query
Frequency of Target Resources (FTR)	300
Physical network size - uniform SoRs	3000,5000,7000,9000 cores
Physical network size - non-uniform SoRs	16500,27500,38500 cores
Homogeneity rate of desired resources	20%, 25%, 33%, 50%, 100%
Uniform SoRs capabilities	10 resources per SoR
Non-uniform SoRs capabilities	normal($\mu = 50, \sigma = 40$)
Desired homogeneous resources/subquery	1 to 6, uniform-SoRs
Desired homogeneous resources/subquery	3 to 18, non-uniform-SoRs
Static subquery dimensions	min=4, max=12
Dynamic subquery dimensions	min=4, max=8

5.4.2.2 Simulation Results

Figs 5.7-a1 and 5.7-a2 depict the experiment results for average discovery latency, and average required number of discovery messages per query, for complex querying in the system with 3000 resources while the SoR capability has uniformly been applied for all the vnode in the system. This means that all SoRs in the system initially have similar capability to supply resources. The SoRs capabilities might also be changed over time depending on their resource release or occupation conditions. In these figures we can see that, as the level of querying complexities in terms of number of required resources and the heterogeneity of the desired resources are increased, the mean discovery latency and discovery cost remain scalable. The experiment results presented in Figs 5.7-bx and 5.8-cx also show the similar behavior for the larger networks with 5000 and 7000 number of resources. However, for the highest complex queries (like the main-queries with the homogeneity rate less than 50% and the number of required resources per sub-query greater than 4), we see that our resource discovery approach gradually becomes worse when we vary the system size from 3000 in Figs 5.7-ax to 5000 in Figs 5.7-bx and 7000 in Figs 5.8-cx. This happens because of several reasons such as, poor stability of SoRs under sub-query requests with high resource demands, disproportion between weak SoRs capabilities and high rated concurrent queries with large resource demands, inefficiency of the HARD's SoR preference mechanism in the lack of SoRs with initially non-uniform capabilities which significantly reduces the degree and scope of competitiveness for SoR preference, and finally exceeding resource demands over resource availability.

QMS providers are able to continuously and dynamically detect and recognize the best possible SoRs for each of their registered resource-type-ids in each moment of time. But these SoRs might become weaker (in terms of number of available resources) over time through reservation of their resources by multiple queries from different requesters. The weak SoRs eventually become unresponsive to the incoming queries, and this will create an extra communication cost to detect the best alternative new SoR as the replacement of the dead SoR. In the experiment results presented in Figs 5.7-ax, 5.7-bx and 5.8-cx, the initial number of available resources supported by each uniform SoR in the system is equal to 10. This means that a fresh SoR dies after successful handling of 10 successive sub-queries with one desired resource, given our assumption for synchronous querying that the discovered resources in each querying iteration would be reserved until end of the iteration. As we increase the number of

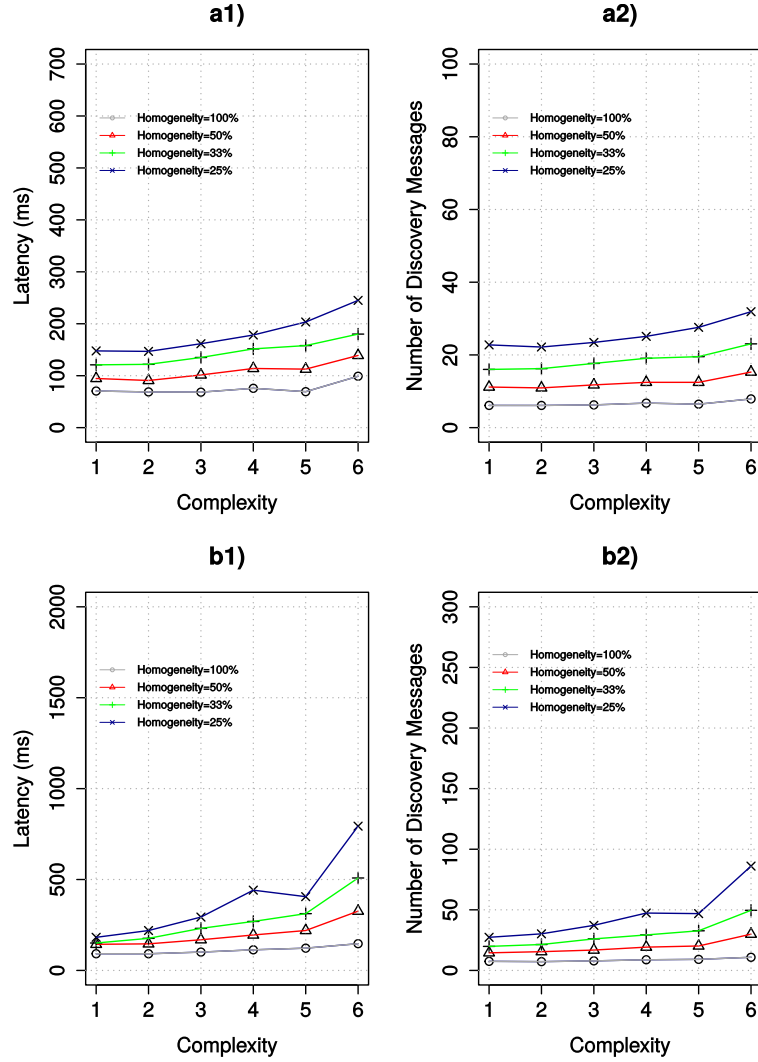


Figure 5.7: Complexity (in terms of heterogeneity of desired resources and number of desired homogeneous resources per sub-query) vs discovery latency and number of discovery messages for different system sizes and different distribution of SoRs capabilities: a1 and a2) uniform SoRs with system size=3000, b1 and b2) uniform SoRs with system size=5000.

desired resources for each sub-query, the instability rate of SoRs in the system is increased. For instance, for sub-queries with 5 desired homogeneous resources, the target SoRs at best die after only 2 (and even less for higher demands) successful queries handled, which leads to high rate of SoRs instability in the system (see Figs 5.8-cx). In these experiments, it can be concluded that our resource discovery approach, for complex querying in synchronous manner with uniform SoRs, can remains scalable, while the amount of demands is atleast less than half of the target SoRs capabilities.

In the aforementioned experiments we assumed that the SoRs capabilities are uniformly distributed, and all the SoRs initially provide equal number of available resources. However in a real jungle computing environment, the SoRs capabilities are not uniform, and in such systems there exist multiple resources with different capabilities and strengths. Along this

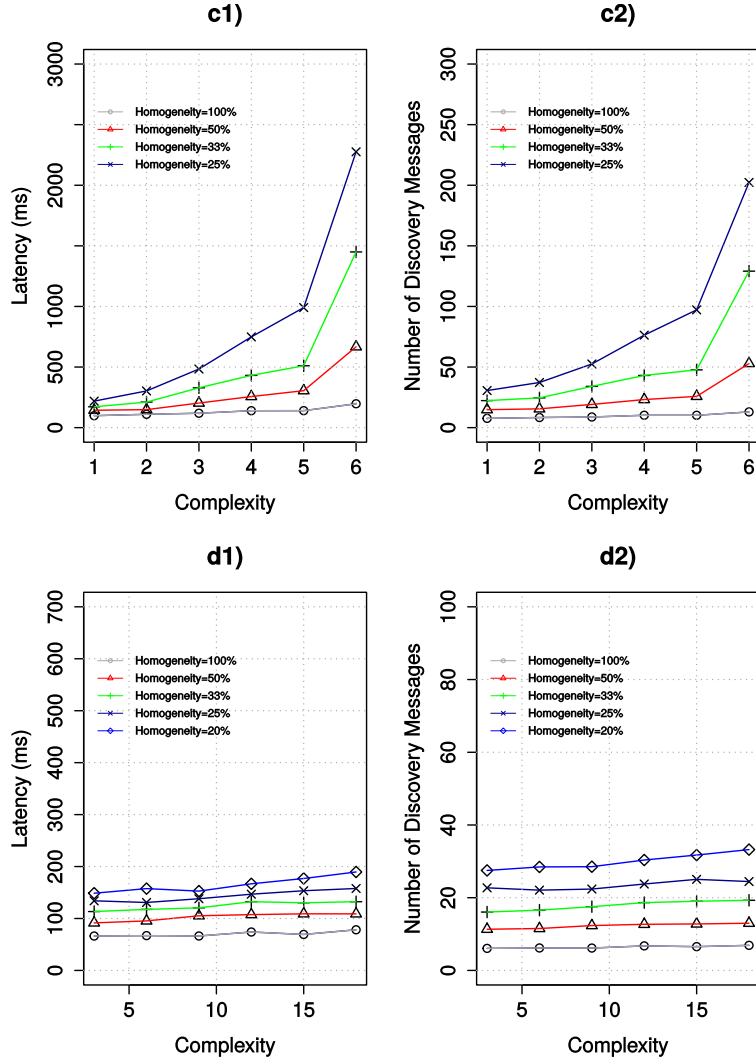


Figure 5.8: Complexity (in terms of heterogeneity of desired resources and number of desired homogeneous resources per sub-query) vs discovery latency and number of discovery messages for different system sizes and different distribution of SoRs capabilities: c1 and c2) uniform SoRs with system size=7000, d1 and d2) non-uniform SoRs with system size=16500.

line we extend our evaluations for non-uniform distribution of SoRs capabilities. The initial number of available resources for each SoR is obtained using a normal distribution with the given mean of 50 and standard deviation of 40 truncated in the range [10,100]. We also increase the level of complexity for the main-queries as well as the system size (see Table 5.2). As we can see in the Figs 5.8-dx, 5.9-ex and 5.9-fx, HARD3 provides significant scalability for discovery latency and discovery cost (i.e., number of transacted discovery messages) when we increase the level of complexity (with respect to the heterogeneity and amount of desired resources for each main-query) for even larger system sizes (16500, 27500 and 38500) and more complex discovery requests (3 to 18 for number of desired resources and 100% to 20% homogeneity rate). Figure 5.10 provides an overview of the results presented in this section.

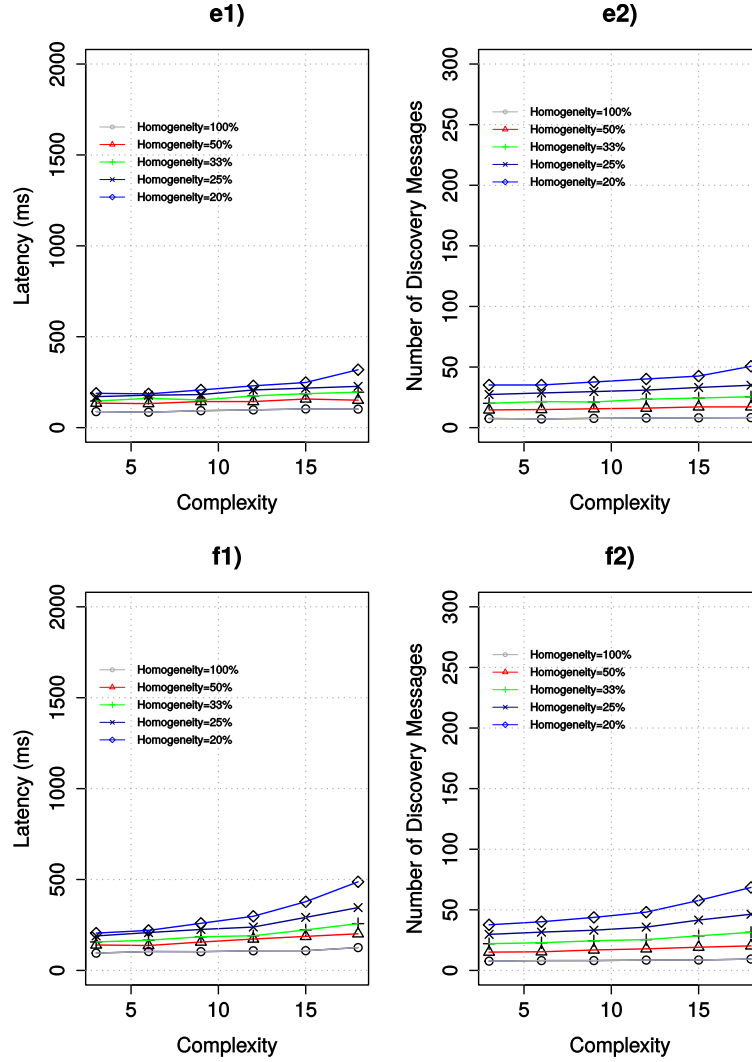


Figure 5.9: Complexity (in terms of heterogeneity of desired resources and number of desired homogeneous resources per sub-query) vs discovery latency and number of discovery messages for different system sizes and different distribution of SoRs capabilities: e1 and e2) non-uniform SoRs with system size=27500, f1 and f2) non-uniform SoRs with system size=38500.

5.4.3 Scalability for Asynchronous Querying

For asynchronous querying in dynamic computing environment, the querying interval for each requester in each iteration is randomly specified by a uniform distribution in the range [2000,6000] ms. Requesters also propagate their successive complex main-queries in the system upon reaching each querying interval. Subsequently, on the successful completion of the resource discovery, the discovered resources would be reserved for the requester in the way that the other concurrent requesters in the system will not be able to discover and get access to the reserved resources until those resources are released by the original requester (i.e., resource occupier). The requesters will release their reserved resources when the execution of the corresponding application is ended. In fact in a dynamic computing environment, resource reservation leads to unexpected unavailability of resources which can be described as the

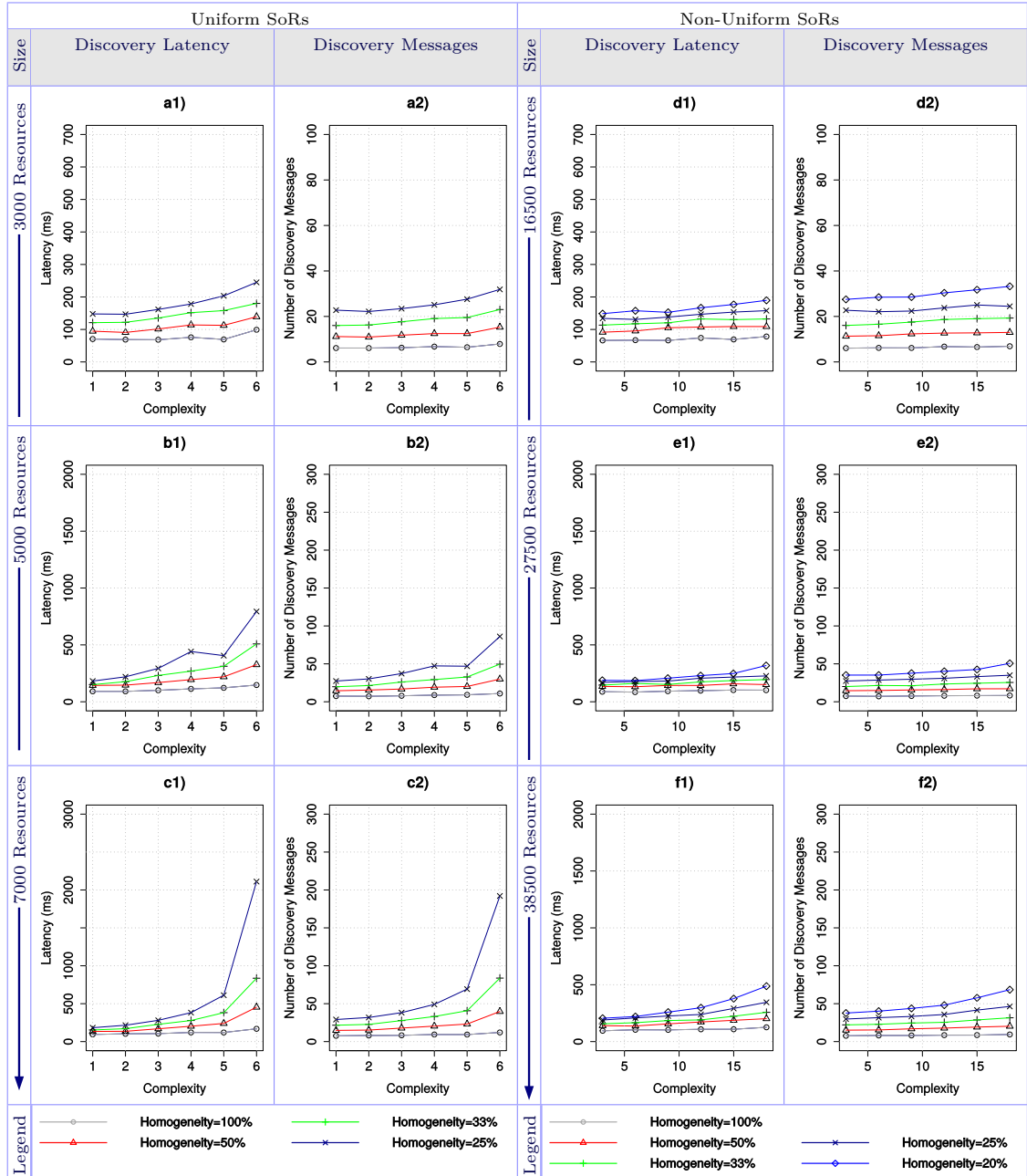


Figure 5.10: An overview of the results: complexity vs discovery latency and number of discovery messages for different system sizes and different distribution of SoRs capabilities.

natural churn. In this section we evaluate the HARD's efficiency and scalability under complex asynchronous querying in dynamic computing environments (i.e., evaluation under natural churn).

5.4.3.1 Simulation Setup

In order to evaluate the HARD3 performance with respect to complex asynchronous querying in dynamic computing environments, we conduct a simulation scenario based on modifications of the previous scenario, as presented in Table 5.3.

Table 5.3: Simulation parameters for asynchronous querying.

Parameter	Values
Complexity of querying	multi-resource/multi-query
Frequency of Target Resources (FTR)	1650
Physical network size - non-uniform SoRS	27500 cores
Execution time (i.e., reservation) uniform	$[i-2000, i=2000*k], k=1-7$
Querying interval by ms	uniform[2000,6000]
Consecutive main-query runs per requester	100
Homogeneity rate of desired resources	33%
Non-uniform SoRs capabilities	normal($\mu = 50, \sigma = 40$)
Desired homogeneous resources/subquery	20, non-uniform-SoRs

5.4.3.2 Simulation Results

Figs 5.11-a1 , 5.11-a2 , 5.11-a3 and 5.11-a4 present the density scatter plots for discovery cost over simulation time (millisecond) for 100 successive main-queries (i.e., discovery requests) per requester in dynamic computing environments with different range of task duration (i.e., application execution) (0,2000] ms, (2000,400] ms, (4000, 6000] ms and (6000, 8000] ms accordingly.

Each data point in the graphs represents the result of a single main-query. The darker points in the graphs (i.e., the high density points) represent states that have a higher probability of occurrence in comparison to the lighter points. As presented in these graphs, the majority of the queries results, particularly the high dense data points, fall on or below the regression line. Similar behavior can also be seen in the Figs 5.12-b1 , 5.12-b2 , 5.12-b3 and 5.12-b4 for discovery latency evaluation over time. This illustrates that HARD3 is highly scalable over time and can efficiently maintains its performance under natural churn, caused by high frequent resource reservations and resource releases in highly dynamic computing environment.

In the aforementioned experiment results, when we vary the distribution range of the task duration from (0,200]ms in Figs 5.11-a1 and 5.12-b1 to (6000, 8000]ms in Figs 5.11-a4 and 5.12-b4, while the range for querying intervals is fixed to [2000,6000]ms, as it is expected, the discovery cost and discovery latency are gradually increased. The reason is that for the larger application execution times, it takes longer for the reserved resources to be released by the original requesters and this leads to higher rate of unavailability for the occupied resources in the system. In this regard, the frequent resources (i.e., very common resources) in the system, might become the rare resources over time with higher cost of discovery. Discovering rare resources might be costly due to the potentially larger search space that is needed to be explored. In addition, the rate of resource unavailability (and becoming rare) would be accelerated particularly when the overall task duration (i.e., application execution time) exceeds the overall querying interval.

The dispersion of the data points is bigger for the graphs with the larger application execution times which shows the impact of resource unavailability, rare resources and resource contentions on the HARD3 performance. The mean hit rates (i.e., the success rate of querying)

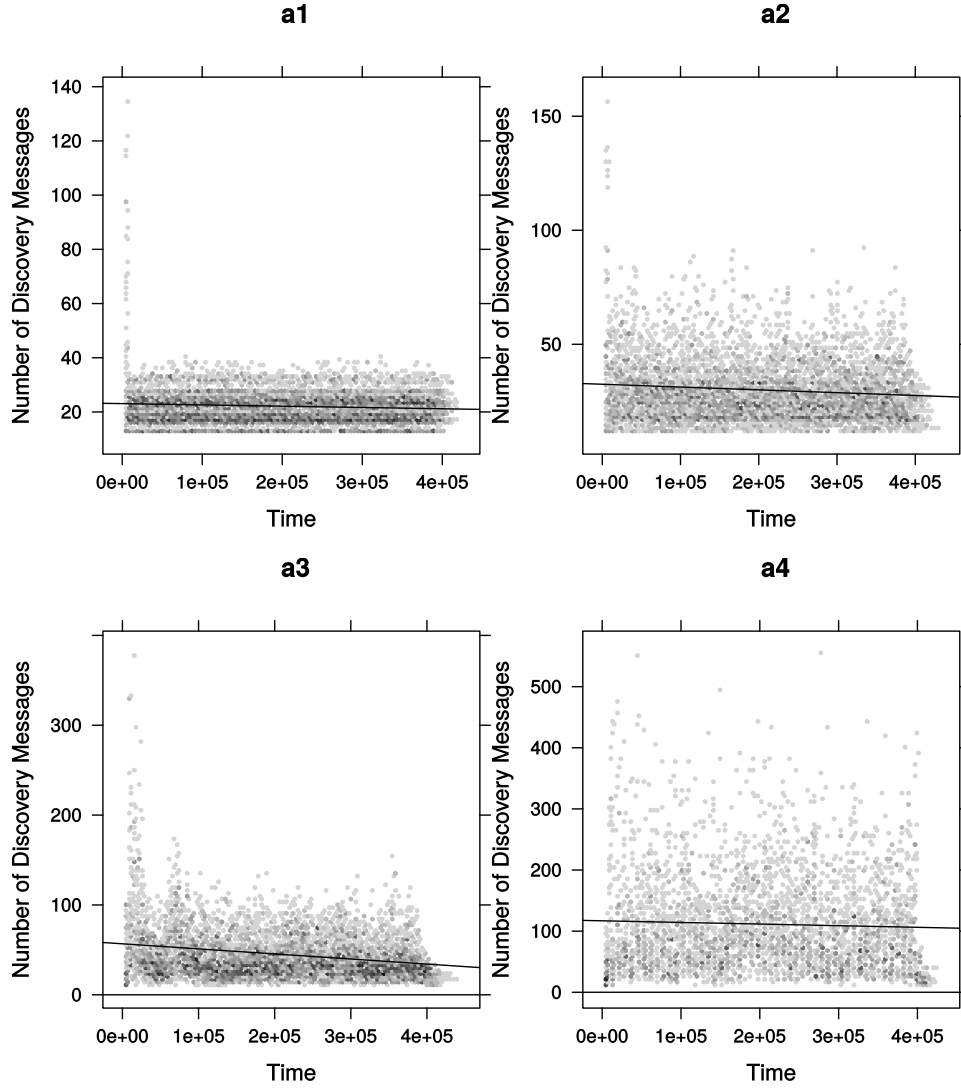


Figure 5.11: Density scatter plot of discovery cost (i.e., number of transacted discovery messages) for fully resolved discovery requests (i.e., hit rate=100%) over time for various application size (i.e., application execution time) while the querying interval is [2000,6000] ms: a1) execution time=(0,2000] ms, a2) execution time=(2000,4000] ms, a3) execution time=(4000,6000] ms, a4) execution time=(6000,8000] ms.

for querying in the experiments presented in Figs 5.11-a1, 5.12-b1, 5.11-a2, 5.12-b2, 5.11-a3 and 5.12-b3 are 100%, which means that all the generated discovery requests by requesters in the system are fully resolved. In Figs 5.11-a4 and 5.12-b4, because of the larger tasks duration, the amount of available resources is decreased in most of the time-slices while the amount of requesters in the systems, issuing new queries, is constant. This can lead to resource contention among the requesters, which in turn reduces the overall hit rate for the discovery requests in the system. The density of the data points in Figs 5.11-a4 and 5.12-b4 is reduced in comparison with other graphs. It means that the number of fully-resolved main-queries is decreased. Thus, as it is shown in Figure 5.13, the mean hit rate is expected to be reduced for

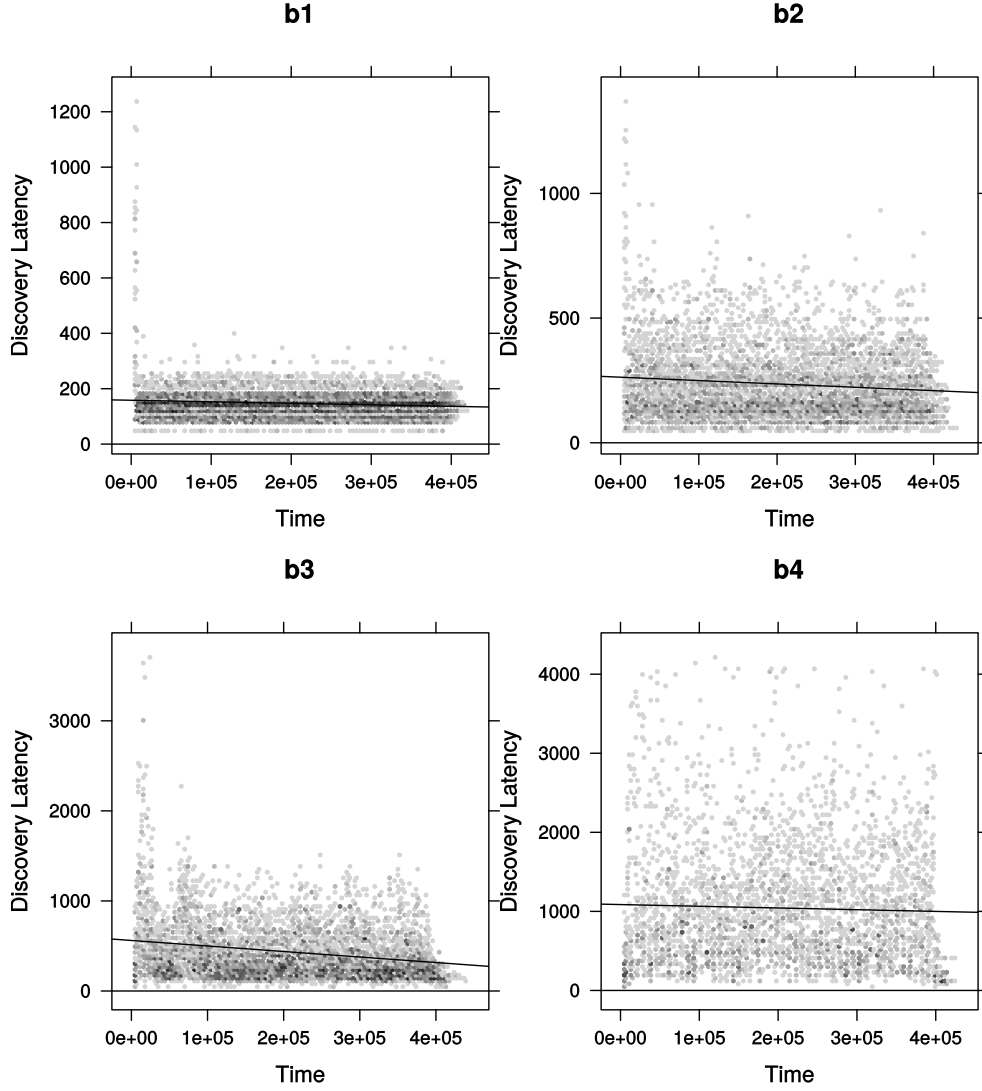


Figure 5.12: Density scatter plot of discovery latency for fully resolved discovery requests (i.e., hit rate=100%) over time for various application size (i.e., application execution time) while the querying interval is $[2000,6000]$ ms: b1) execution time= $(0,2000]$ ms, b2) execution time= $(2000,4000]$ ms, b3) execution time= $(4000,6000]$ ms, b4) execution time= $(6000,8000]$ ms.

the task duration= $(6000,8000]$ ms.

Figure 5.13 illustrates the hit rate for querying in the system with different task duration. It shows that the mean hit rate is decreased while we increase the tasks duration, and the hit rate is equal to 100% when the overall execution time is less than querying interval. In fact HARD3 is able to precisely and successfully discover all the desired resources for the discovery requests without any specific limitations. The hit rate reduction only happens when there are not enough available matched resources for all the concurrent discovery requests in the system in a moment of time, which leads to resource contentions. But the HARD's performance and efficiency in itself is completely isolated from the external conditions such as

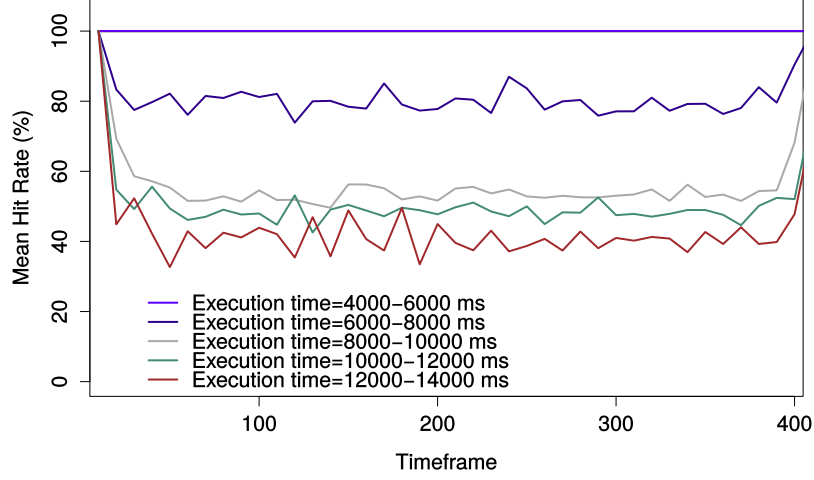


Figure 5.13: Mean hit rate for queries within the time-frames (each time-frame=10000 ms) for different application execution times in asynchronous querying with system-size=27500 resources.

resource unavailability and resource contention.

5.4.4 Comparison with Different Proposals

In this section, we present the simulation results which demonstrate the performance of our resource discovery scheme (i.e., HARD) in comparison to other alternative approaches. For comparison, we simulate our discovery scheme, HARD, in conjunction with three generic hybrid distributed approaches: a 2-layered hybrid DHT, learning-based and partial random-walk based discovery (PRW2), a 2-layered hybrid DHT and full random-walk based discovery (FRW2) and a 2-layered hybrid broad-cast and full random-walk based discovery (BRW2). We also simulate two versions of HARD, HARD3 and a 2-layered instantiation of HARD (HARD2) for assessing the impact of hierarchy and our proposed layer-based query resolution methods on the HARD's performance. We already discussed and presented the details of HARD3 in the previous sections. HARD2 is a two-layered non-anycast based implementation of HARD, which is identical to HARD3 except that it doesn't support SN layer and SQMS providers and instead it resolves any SN-dependent queries (e.g., $\langle n_{ln}.n_{an}.c_{sn} \rangle$) by extending the native probability mechanism of HARD3 to support the layer_{sn} resource information within layer_{an}.

Similar to HARD2, PRW2, FRW2 and BRW2 are organized on top of two-layered (i.e., leaf-node layer and aggregate-node layer) distributed hierarchies. PRW2 and FRW2 leverage the same Chord based DHT method which is used in HARD3 in the leaf-node layer while they provide different query forwarding methods in the aggregate-node layer. PRW2 uses both probability and random-walk method to guide queries in layer_{an}. In this approach, distributed probability tables in the system only process the query results with respect to the resource information in layer_{ln} and layer_{an}. PRW2 might behave similar to HARD2 for non-SN-dependent queries (e.g., $\langle c_{ln}.c_{an}.n_{sn} \rangle$), but for SN-dependent queries the selection

of the forwarding destination node is partially random, since the probability tables do not actually care about the required resource conditions in the super-node layer for these queries.

PRW2 is comparable to our approach (i.e., HARD3) in the sense that it creates clusters on top of the unstructured network. It also provides similarity to some well-known request propagation strategies in the literature such as the shortcut, random walk, learning-based, best-neighbor, learning-based+best-neighbor methods. These methods have been used in many popular resource discovery systems and applications [433–438]. For example in Iamnitchi et al [439] a fully decentralized discovery approach is proposed, which is based on publish/-subscription of the resource information on some specific nodes in the virtual organization. Learning-based and also random-walk methods are used to propagate the queries among the server nodes. Our approach (and PRW2) are not based on publish/subscription since it has costs in terms of network traffic, processing, and storage needs for periodical updating and the maintenance of resource information particularly in high dynamic environment. On the other hand, our probability mechanism is comparable to, or even better than, learning-based strategies. In the learning-based method, nodes learn from experience by recording the requests answered by other nodes (i.e., by caching the results of successful queries). A request is forwarded to the node that has answered similar queries previously [440]. This strategy becomes inefficient when the system size, dynamicity and heterogeneity of resources/queries increases due to the larger memory requirements to maintain the query results and unavailability of the pre-discovered resources. But in our proposed probability mechanism, the statistical information about all the transacted queries by each peer are aggregated in the fixed-size DPTs regardless of the successfulness of the queries. In addition, by leveraging techniques such as dynamic best SoR detection, low-resource nodes and resource unavailability detection and various situation-based policies and updating strategies (e.g., shortest path, latency-aware and attribute-based updating) our proposed probability method provides better accuracy and efficiency.

Unlike PRW2, FRW2 employs a fully single random-walk method to guide all type of queries in aggregate-node layer. Random-walk is a common query forwarding method, which is originally proposed in the literature to alleviate the excessive traffic problem caused by flooding [441], and to deal with the traffic/coverage trade-off. Random-walk is used in many distributed resource discovery applications such as Gnutella [442, 443], Iamnitchi et al [439] and [444–448].

BRW2 or Broad-Walk is a hybrid two layered approach which uses broadcast-based query propagation method [449, 450] in the leaf-node layer and the random-walk forwarding in the aggregate-node layer. In continuation of this section we explain the details of our simulation setup and we discuss the comparison results.

5.4.4.1 Simulation Setup

Using our self-organized clustering algorithm we simulate the aforementioned discovery approaches, on top of either two-layered or three-layered distributed hierarchies. Similar to the previous scenarios, we simulate dynamic computing environment containing various number of computing resources, in which a constant number of resources (i.e., requesters) simultaneously issue the discovery requests to the system. The time interval between each pair of consecutive queries issued by a requester is defined by an exponential distribution. We also assume that each requester issues 10 consecutive resource requests to the system over the simulation time. The discovered resources will be reserved for each discovery request. The reserved resources

for each process will be released after execution time period which is defined by a Weibull distribution. We execute 10 queries per requester for each system size, each one originating from a uniformly chosen source-node. Each experiment for each system-size is repeated for 100 runs with different topology parameters. All the queries are identical and represent the queries of type $\langle c_{ln}, c_{an}, c_{sn} \rangle$. Each requester is willing to find required resources for a process containing three thread-groups with different resource requirements. Table 5.4 presents more details of simulation parameters for our evaluation.

Table 5.4: Simulation parameters for performance compression.

Parameter	Values
Physical network size	5500-55000 cores
Interconnect topology	Mesh/Torus
Network topology	Random
Interconnect channel datarate	50Gbps
Network channel	100 Mbps
Desired Resources for each Request	3x20
Homogeneity rate of desired resources	33%
Frequency of Target Resources (FTR)	1650
Process Duration by sec	Weibull($\lambda=3.58, k=2.40$)
Querying Interval by ms	Exponential($\beta=4000$)
Consecutive Query Runs per Requesters	10
Rate of Requesters	1%

5.4.4.2 Simulation Results

In the first test, we perform experiments to measure the average number of required messages, and the average latency (by milliseconds) per discovery request for HARD3, HARD2 and other approaches. The PRW2, FRW2 and BRW2 with the same topology, simulation parameters and conditions are used as alternative reference works. Figs 5.14-a, 5.14-b and 5.14-c show plots of the average discovery messages, and the average discovery latency per query, as a function of the number of computing resources in the system (i.e., system size). Figure 5.14-b demonstrates the results presented in the Figure 5.14-a with better resolution (without BRW2).

In Figure 5.14-a, we observe that the average required number of discovery messages per query for BRW2 is much larger than the other approaches, while in Figure 5.14-c the average latency of BRW2 is close to FRW2, PRW2 and HARD2. This means that BRW2 significantly generates more discovery traffic in comparison to others, due to the heavy cost of broadcasting in $layer_{ln}$. The queries in BRW2 are guided in the aggregate-node layer by being forwarded to a non-visited single random neighboring aggregate-node. Upon arrival of a query in an aggregate-node, if that node fits the query conditions (i.e., c_{an}) for the current layer (i.e., $layer_{an}$) the query is broadcasted to all the leaf-node members of the current aggregate-node, otherwise it is forwarded further in the network using random-walk. Broadcasting results in increased traffic, but as seen in Figure 5.14-c, this could provide reasonable response time for queries, since the aggregate-node inquires all of its leaf-node members in parallel. The response time for BRW2 is approximately close to the results for FRW2, PRW2 and even HARD2.

Figs 5.14-b and 5.14-c show that our approach, HARD3, provides the highest performance and scalability among others for both discovery traffic (i.e., average number of discovery messages propagated during a search), and latency when varying the number of resources

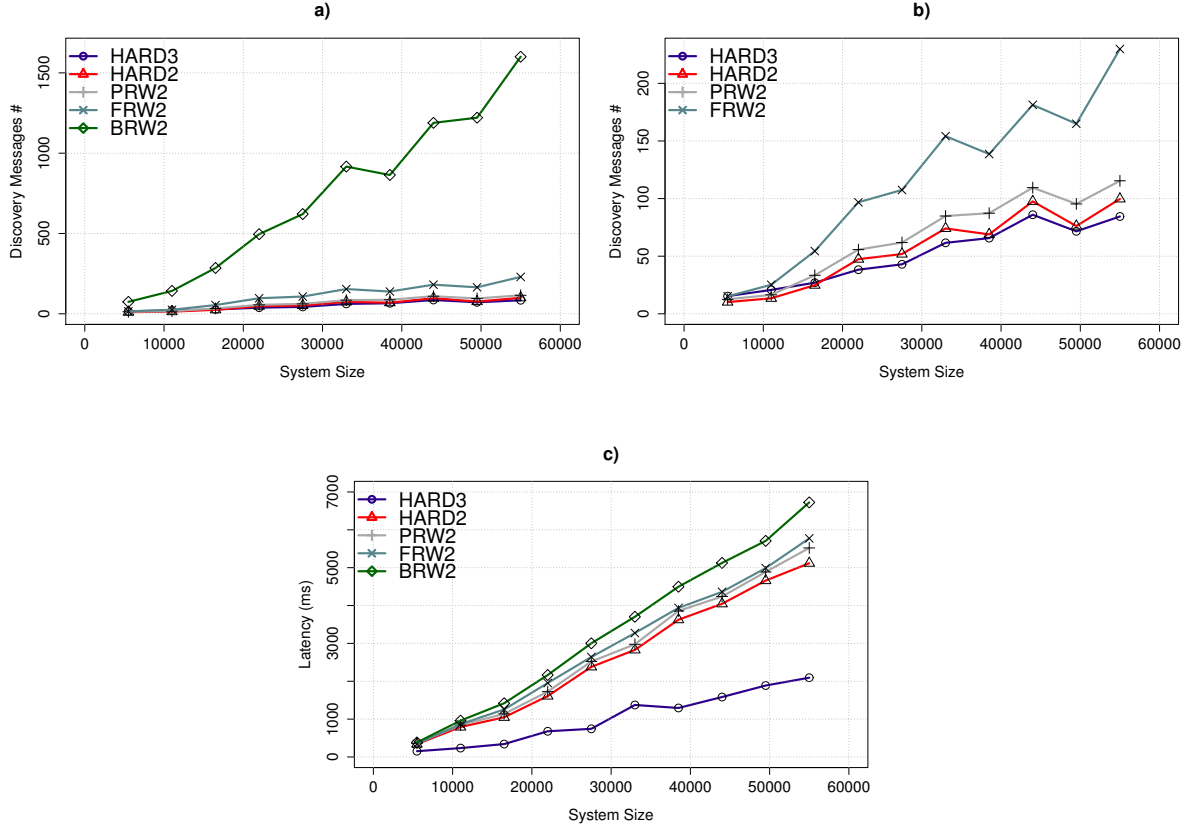


Figure 5.14: Comparison between HARD3 and other alternative approaches: a, b & c) average number of required discovery messages and discovery latency per discovery request for different strategies and system sizes.

in the system from 5500 to 55000 resources. This is particularly significant for the query's response time (latency) since other approaches, such as HARD2 and PRW2, also provide close results in terms of number of discovery messages. HARD3 efficiently divides the exploring space to the anycast groups in a way that, queries with c_{sn} requirements are only propagated among the SQMS providers whose specifications in $layer_{sn}$ fulfill the c_{sn} conditions of the given query essentially. In comparison to HARD2, this strategy leads to a significant reduction in the response time of HARD3 while its discovery traffic is also slightly decreased. As we already discussed, HARD3 is the enhancement of HARD2 by leveraging our proposed anycast forwarding mechanism in an extra layer which is called $layer_{sn}$. The presented results for HARD2 and HARD3 in Figs 5.14-b and 5.14-c also prove that increasing the level of hierarchy along with the implementation of an efficient adaptive corresponding query processing method improves the overall performance of our discovery system. Another factor contributing to HARD3's overall performance is that HARD3 controls the discovery procedure in a more intelligent way which saves much unnecessary message cost. Moreover due to the anycast nature of HARD3, SQMS providers are able to effectively guide the given queries to the closest qualified SQMS provider in the system which results in a significant reduction in the discovery latency for the queries in the system (see Figure 5.14-c).

From Figure 5.14-b, we can also see that, PRW2 generates larger number of discovery messages per query than HARD2 and HARD3 because of its partial random-walk query forwarding mechanism in $layer_{an}$. In fact, PRW2 provides an efficient probability mechanism (similar to HARD2) to guide queries in $layer_{an}$ to the potential matched resources in the system. But this probability mechanism becomes inefficient for processing the queries with c_{sn} requirements because the DPTs in PRW2 do not consider the c_{sn} requirements of the given queries in order to statistically estimate the target aggregate-node for query forwarding. This leads to a sort of partial random-walk for the SN-dependent queries (i.e., for the queries that $c_{sn} <> n_{sn}$). But for the other types of queries (e.g., $< c_{ln}.c_{an}.n_{sn} >$), which are not considered in our evaluation in this section, PRW2 is expected to behave identically to HARD2.

Figure 5.14-b illustrates that FRW2 provides a lower performance with respect to discovery overhead compared to PRW2 while they exhibit almost similar behavior for discovery latency as shown in Figure 5.14-c. This is due to the fact that the mechanism for query resolution in $layer_{an}$ of FRW2 is fully based on random-walk method. This means that the queries in $layer_{an}$ are forwarded to a uniformly random selected neighboring aggregate-node in the system which is not yet visited. Since the next-node selection strategy is completely random-based the number of required traversed discovery messages for resolving a query would get more compared to the approaches benefiting a type of estimation-based strategy.

In the next experiments we analyze the dynamic behavior of the above-discussed approaches for various system size over time. The simulation results for both average generated discovery messages and average discovery latency per request per time-frame (1000 ms) for HARD3, PRW2 and FRW2 are depicted in Figs 5.15-a1/a2, 5.15-b1/b2 and 5.15-c1/c2 respectively. These results lead to the following conclusions:

(a) The overall variation and fluctuation in the query results (in terms of discovery overhead) per time-frame for HARD3 is less than PRW2 and FRW2. This means that HARD3 provides better scalability over time with respect to discovery overhead.

(b) The figures that demonstrate the amount of discovery overhead per time-frame for various system size for HARD3 provide relatively better approximation of the overlap than the corresponding charts for PRW2 and FRW2. This means that HARD3 outperforms the other approaches with respect to scalability (in terms of discovery overhead) when varying the number of computing resources in the network from 22000 to 55000. This behavior can also be seen in the figures illustrating the average discovery latency per time-frame for various number of computing resources. Thus, we can conclude that HARD3 provides better scalability (in terms of discovery latency) for different system sizes.

(c) The latency figures in all the approaches approximately provide similar steady behavior except for a portion of time when the requesters gradually start or stop sending series of discovery requests to the network. This unsteadiness happens because we assumed that at the beginning of the simulation all the computing resources in the system are free (not reserved), and that is how the initial queries for each requester would be able to discover their desired resources in a shorter time. As time proceeds, the total number of reserved resources in the network to execute the waiting processes of the requesters increases which leads to increasing the response time for the new queries. The requesters will release the reserved resources for each of their processes after the execution terminates. This is the reason for the dramatical increment of the response time of the queries at the beginning of the experiments. Similarly, the response time for the queries decreases dramatically at a portion of time at the end of the experiment, when the requesters in the system gradually stop sending new queries after issuing fixed number of successive queries. A requester may stop querying earlier or later than

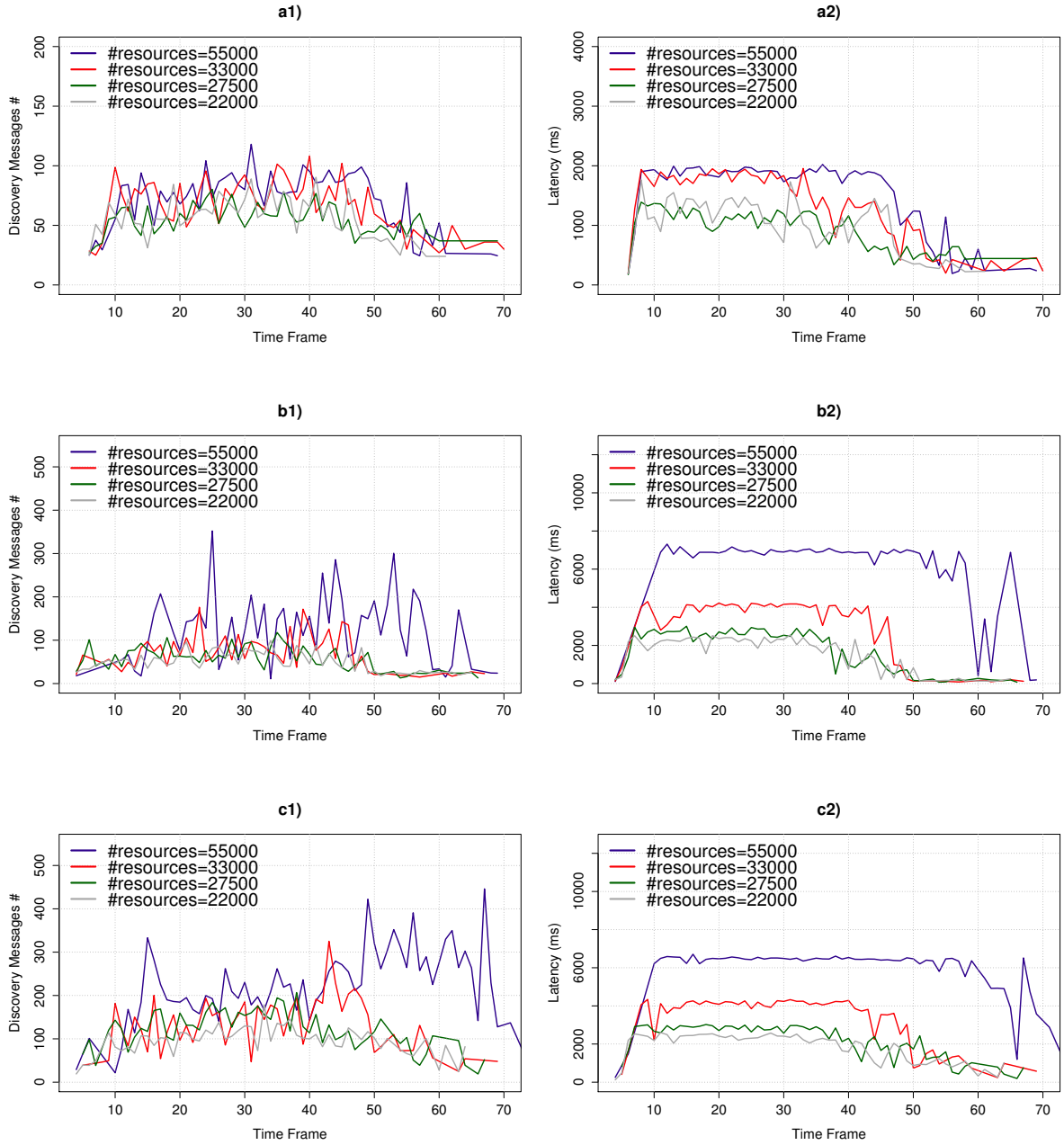


Figure 5.15: Comparison between HARD3 and other alternative approaches: a1 & a2) mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system sizes for HARD3, b1 & b2) mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system sizes for PRW2, c1 & c2) mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system sizes for FRW2.

other requesters depending on its various querying interval time before generating each new query. With respect to the aforementioned behavior of the latency charts in our experiments

we can conclude that all the approaches provide scalability for discovery latency over time. But HARD3 shows better scalability (in terms of latency) than others for different system sizes as elaborated earlier.

In the next simulation, we measure the overall discovery load per node (i.e., vnode) during the querying period (60000-80000 ms), which is the duration of time that the requesters propagate a constant number of successive queries across the network in a parallel manner. The querying period ends when the request-initiator corresponding to the last query is replied. The discovery load is the average number of transacted discovery messages by each vnode during the querying period. We also measure the overlay load per node (i.e., the average number of transacted overlay messages per node to create the underlying hierarchical system overlay) and Transmitted Discovery Data (TDD) per node (i.e., the average amount of transmitted discovery data per node during the querying period).

Figs 5.16-ax, 5.16-b and 5.16-c depict the discovery load per node, overlay load per node and TDD per node respectively, as the function of system size for different approaches. As we can see in Figs 5.16-ax and 5.16-c, HARD3 shows better performance compared to other solutions. However Figure 5.16-b shows that the overlay cost (in terms of the average transacted overlay messages per node) to establish the underlying overlay for HARD3 is relatively larger than in others.

As we already discussed, all approaches employ the same multistage hierarchical overlay algorithm [451] to implement either two-layered or three-layered structure. The overlay cost to create the structures with higher level of hierarchy like HARD3-overlay, which has three layers is bigger than the structures with lower levels of hierarchy, such as the underlying overlay of HARD2, PRW2, FRW2 and BRW2. The overlay cost for BRW2 is the least among the other two-layered based approaches because it provides a non-DHT based approach for query processing in $layer_{ln}$ unlike HARD2, PRW2 and FRW2. BRW2 does not require the creation of a DHT structure within the $layer_{ln}$ which has some extra overlay cost.

5.4.5 Other Features

Our proposed discovery approach provides a set of important functionalities/features in order to support complex and flexible querying within large scale distributed systems. In this section, we briefly compare our approach with some discovery examples in the literature in terms of querying features. These approaches include SWORD [57], Node-Wiz [348], MDS-4 [14], MatchTree [62], CycloidGrid [64] and OntoSum [192]. Table 5.5 presents the result of our qualitative assessment. Following, we provide a short description for some of the features, used in the comparison.

Nearest Neighbor Query is a discovery capability to provide a list of discovered resources considering the priority of the closer neighbors. Similar Matching is the ability to find a similar match for a query with respect to query conditions. Resource Graph Discovery is the capability to discover a graph of resources considering both individual characteristics and interconnecting properties of resources. Multi-Dimensional and Range Querying is the capability of the discovery system to process queries containing multiple attributes either dynamic or static within specific ranges of values. Thread-level discovery specifies the capability of the discovery system to deal with the query conditions in thread-level (i.e., resource requirements for each thread in a process).

Overall, HARD3 seems to present the best solution for JCS, with the most complete set of features. Even of HARD3 does not provide "Nearest Neighbor Query", it is proximity-aware,

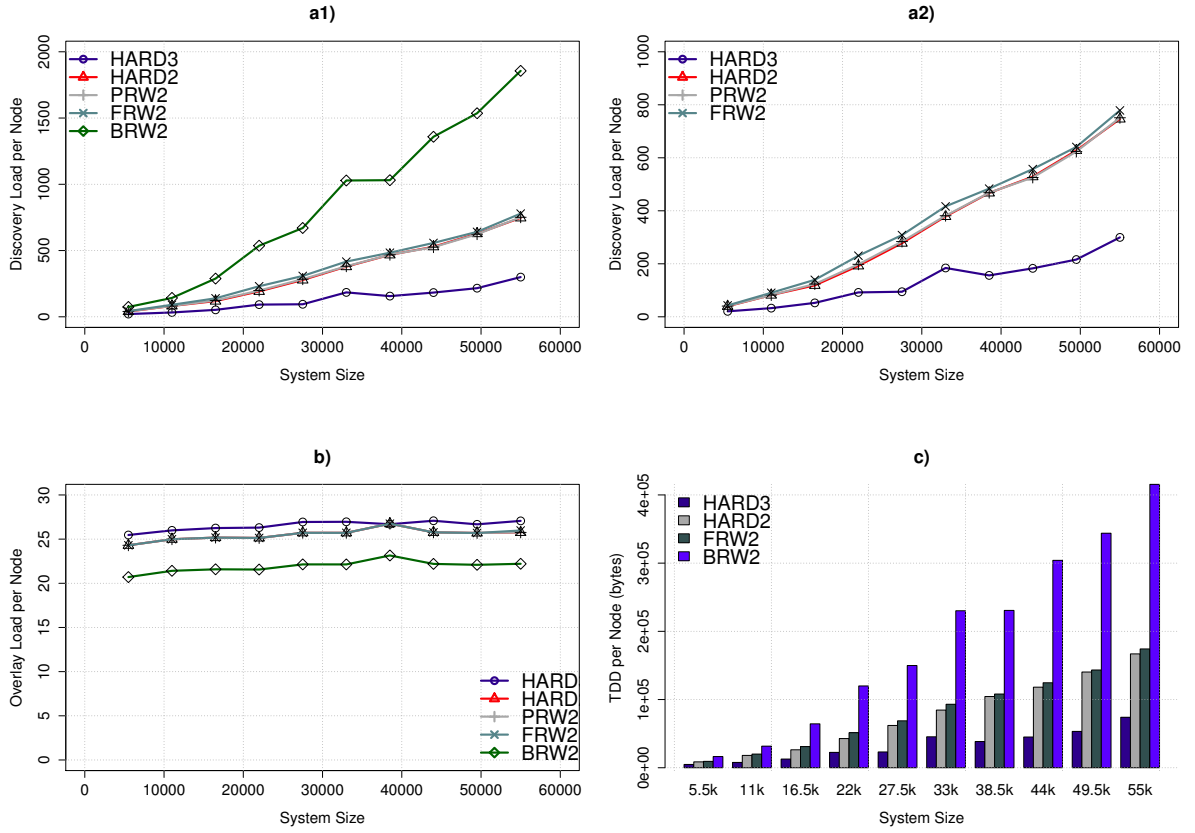


Figure 5.16: Comparison between HARD3 and other alternative approaches: a1) average discovery load (number of transmitted discovery messages) per node (i.e., *vnode*) during simulation time (60000-80000 ms) for various strategies and system sizes, a2) provides better resolution of the results presented in a1 for different strategies excluding BRW2, b) average overlay load (number of transmitted messages for overlay construction) per node during simulation time (60000-80000 ms) for various strategies and system sizes, c) average transited discovery data per node during simulation time (60000-80000 ms) for various strategies and system sizes.

which copes with this issue. Also, HARD3 is inherently providing "Load-Balancing", due to the probability aspects of the search algorithms.

Table 5.5: An overall comparison between HARD3 and examples of other discovery approaches in terms of querying features.

Functionalities	SWORD	Node-Wiz	MDS-4	MatchTree	CycloidGrid	OntoSum	HARD3
Load Balance		✓		✓	✓		
Fault Tolerance		✓	✓	✓			✓
Self-Organization	✓	✓		✓		✓	✓
Range Query	✓	✓		✓			✓
Similar Matching							✓
Multi-Dimensional Query	✓	✓		✓			✓
Resource Graph Discovery	✓						✓
Nearest Neighbor Query	✓					✓	
Proximity-Awareness					✓		✓
Resource Reservation			✓		✓		✓
Semantic-Awareness						✓	
Thread-Level Discovery							✓

5.5 ElCore Evaluation

In this section, we evaluate the performance of our proposed resource management scheme with respect to different evaluation criteria. We use simulation instead of experiment on the real large-scale computing infrastructures (e.g., PlanetLab, TACC, Oak Ridge, BSC, GENCI and public Cloud providers like Amazon and Google) due to low cost and flexibility of the simulation to design, development and evaluation of the new algorithms as well as providing full control over system behavior and evaluation scenarios. Furthermore, the real infrastructures generally provide limitations to explore the design space particularly for scalable performance and large-scale evaluation.

In general, one of the traditional evaluation criteria, is resource utilization, which demonstrates how the resources in terms of computation and communication capabilities (e.g., CPU usage, Memory usage, Network Bandwidth, etc.,) have been efficiently utilized, in order to distribute and manage the workload. Resource utilization as an evaluation criteria has been used for many resource management systems specifically for virtualization based resource management systems or Cloud platforms available today.

In our evaluation, we decided to use the Resource Allocation Accuracy (RAA) as the main criteria to evaluate our work instead of Resource Utilization. RAA indicates how much a set of allocated resources for a process fulfills the original resource requirements, emphasized by the PM requester.

In our resource management scheme, we assume that PM would dynamically be able to precisely identify the resource requirements for each process and consequently the next level RMC would be responsible for finding and reserving the optimum set of qualified resources, which will be allocated to the relevant threads by PM, where the scheduler will be invoked to run the process accordingly.

We note that our assumption of dynamic identification of resource requirements for given applications/workloads (by PMs), may not be fully practical in current manycore systems (or implemented in current operating systems), but it is becoming a practice in nowadays Cloud computing technologies [452–463]. We argue that current Cloud computing systems are a potential ancestor of the future manycore systems (Intel 48-Core “Single-Chip Cloud Computer” [464] is an example of this trend). Since the target computing environments of ElCore is the future manycore systems, our assumption is reasonable; this is further supported by the concepts developed in the context of S[o]OS (which is an example of distributed operation systems, concerning the requirements of future manycore systems) [465]. In other

words, S[o]OS is not for today, but we can see the ancestors of S[o]OS on current cloud systems, where the service demand (or the resources needed to run a job) is estimated prior to VM allocation and job execution [452–456].

Due to the above assumption, and the fact that the efficient mapping between the RMC-offered resources and the PM resource requests has significant impact on the whole system performance, we consider RAA as one of the most important performance criteria for managing resources in future large-scale manycore environments (with presence of high diversity of resources and applications) since it fully depends on the quality of resource management component, while resource utilization very much depends on the specific dynamicity of the requests being processed as well.

As it is shown in Equation 5.1, RAA for each query can be measured by calculating the ratio of the fully satisfied conditions to the total number of query conditions (in terms of the required computing attributes for each single resource, and the desired inter-resource communication properties).

$$RAA = \frac{\gamma + \sum_i^g \left(\sum_j^{m_i} (\phi_j) + \tau_i \right)}{L + \sum_i^g (n_i \rho_i + l_i)} \times 100 \quad (5.1)$$

Here, ρ_i is the number of desired (computing) attributes for each requested resource in a sub-query for a group of homogeneous resources (i is the group index); n_i and m_i are respectively the number of requested resources and discovered resources for each group i ; l_i and τ_i are respectively the number of dependency links and qualified links for each group; L and γ are respectively the number of inter-group links (communication conditions) and qualified links for all groups; g is the number of resource-groups (number of sub-queries for the query) and ϕ_j is the number of qualified attributes for a discovered resource (j is the resource-index).

5.5.1 Simulation Setup

To do our evaluation, we developed a simulation platform, based on OMNET++ (similar to the way that is presented in [466]), which is able to simulate manycore environments (up to 55000 cores in different chips and nodes), focusing on communication aspects (i.e., communications between cores, chips and nodes). Using our simulator, we have simulated a manycore networked environment containing 2000 computing resources (i.e, processing cores), with 6 different types of processing resources. Each of these types have their own specific computation properties in terms of core clock rate, cache size, cach line, ALU properties and functionalities, memory bandwidth, etc. For instance, the processor frequency for resource type A, B, C, D, E and F are 2.53 GHz, 3.6 GHz, 1.6 GHz, 2.8 GHz, 1.2 GHz and 2.4 GHz accordingly. We conduct our simulation in a way to produce 20 percent of the resources as distributed homogeneous resources of a specific type-A which later will be required for several different processing scenarios. The reason for adjusting the amount of a certain resource-type in the system (as the target resources for the queries) is to facilitate measuring the value of Maximum Reachable Accuracy (MRA) in the different experiments, but in fact it does not impact the generality of our evaluation because the resource management procedure is completely independent from the frequency of the target resources.

The type of resources, as well as their distribution in the system, is random. To do this, in the first step, we randomly order all processors (of different nodes) in a queue. We create

an array of pairs (tx, ty) which contains all different two-combinations from the given set of types $S=\{A, B, \dots, F\}$. We iterate through the array, and for each iteration, we dequeue a processor from the queue and proceed to randomly specify the type of tx or ty for each of its cores. The iteration is repeated and the process continues until the queue becomes empty. As a result, each processor in the system, will consist of maximum two different types of cores (heterogeneous cores). In the next step, we control and regulate the desired amount of cores of a given specific type (type-A) through modifying the type of cores from type-A to ty or from ty to type-A, in the processors that contain cores with type-A ($tx=\text{type-A}$).

Figure 5.17 shows the overall architecture of the simulated manycore system that we use for the evaluation in this section. The cores of each processor are connected through a three-dimensional mesh interconnect topology (with datarate of 50Gbps) which, in our simulation scenario, is the same as three-dimensional torus. Each core has its own dedicated L1 and L2 cache that are not shared with any other cores. We do not simulate cache coherency protocols, as we use message-passing for inter-core and inter-chip communication. The routing among cores is performed through a variation of Dimension Order Routing algorithm [467] where packets are routed to the correct position along higher dimensions (x or y) before being routed along lower dimensions (y or z). Furthermore, the processors of each node are connected through a high speed bus with datarate of 20 Gbps and we construct a network (with bandwidth of 100Mbps) based on a random network topology, a connected graph with $\beta = 62.5\%$ that indicates the ratio of the number of nodes to the number of links in the network graph.

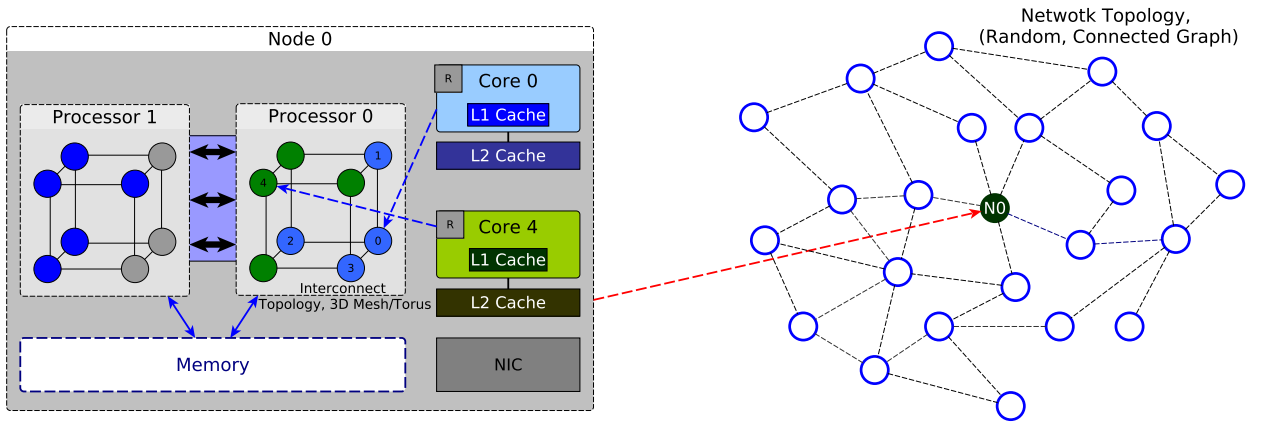


Figure 5.17: Manycore Simulation - System Architecture

We define 4 settings for the processing scenarios. Every setting is evaluated in 10 different runs. In each run the network topology parameters will be randomly changed according to our predefined simulation parameters (see Table 5.6).

In the first setting we assume that PMs will generate 1000 resource requests in fixed intervals (4000 ms) for different processes of the same type (i.e., in each interval, only a single request is generated in the entire system). The predefined process in this setting includes 2 threads that need 2 resources of type-A with specific communication requirements between those resources (maximum $150 \mu s$ for each dependency link). The other settings are similar to the first setting except for the type of processes. However, for the sake of simplicity, the

Table 5.6: Simulation Parameters for Accuracy Evaluation

Parameter	Values	Description
Physical network size	2000 cores	Total number of resources in the system (i.e., total number of cores)
Number of network nodes	125 nodes	Total number of connected network nodes in the system
Number of tiles (CPUs) per node	2 processors	Number of processors per network node
Number of cores per CPU	8 cores	Each processor contains maximum two different types of cores
Interconnect topology	3-Dimensional Mesh/Torus	Topology among cores of each processor
Network topology	Random (connected graph, $\beta=62.5\%$)	Topology among network nodes, β indicates ratio of #nodes to #links in the network graph
Interconnect channel data rate	50Gbps	Inter-core communication
Routing type	DOR	Dimension Order Routing algorithm for routing among cores
Bus speed	20 Gbps	Inter-processor communication
Network channel	100 Mbps	Inter-node communication
Vnode clustering	$\eta=8$, $\lambda=100\text{ns}$	Clustering is limited to include cores inside multiple chips of a single common network node
Desired inter-resource latency	$(0, 150\mu\text{s}]$, maximum 150 μs	The maximum inter-resource latency required for each query
Process Duration by sec	Weibull($\lambda=3.58, k=2.40$)	Execution time for each process through Weibull distribution, λ is the scale parameter and k is the shape parameter.

type of requested resources in all of these settings are homogeneous (type-A). The difference between the processes in the aforementioned settings is the number of requested resources, as well as the level of dependency between the threads. We also assume that the required resource for each thread is a single processing resource (i.e., a core). The required number of resources for each process in settings 2, 3 and 4 are 3, 5 and 9 while the threads dependency constraints are applied for 2, 4 and 8 communication links accordingly (see Figure 5.18).

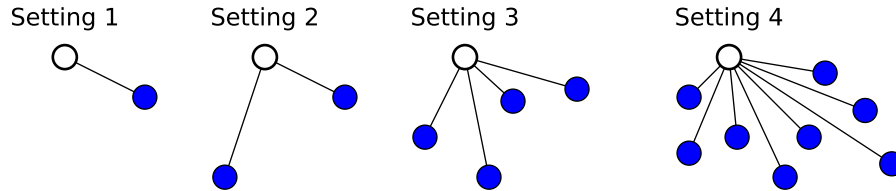


Figure 5.18: Dependency graph for different settings

In each run, after a query is finished, the matched resources will be allocated to the corresponding process. The allocated resources will be released after a period of execution time (termination of the process, see Table 5.6). In our simulation we assume that the released resources will be immediately transferred from the free-list to the free-ownership-list. For each resolved query, the simulator records information including query description, query results and the current status of the network (system) graph as a trace. Upon completion of simulation, the simulator individually analyses the generated traces for each run in order to calculate values of RAA and Maximum Reachable Accuracy (MRA) for each query. MRA is measured, similar to RAA (as it is presented in Equation 5.1). But, unlike RAA, MRA uses the parameters of the ideal query results, instead of the real query results achieved by our proposed resource management approach. To do this, the simulator statically evaluates the query conditions for each potential set of results in the network graph (provided by a trace) and concludes with the most optimal results attainable for the given query.

5.5.2 Resource Allocation Accuracy (Mapping Quality)

Figure 5.19 demonstrates the simulation results for each of the defined settings for different runs. The linear plots above the bar charts in each graph represent the variation of the value for MRA which is dependent on network topology and random distribution of resources.

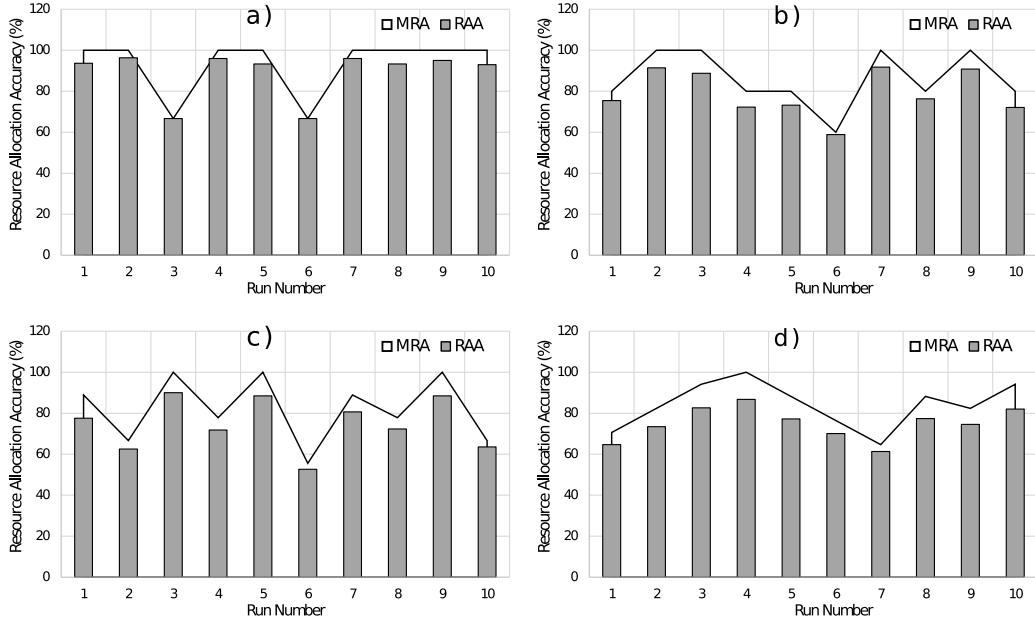


Figure 5.19: RAA ratio for different settings and network topology parameters: a) Setting 1: the processes with 2 threads b) Setting 2: the processes with 3 threads c) Setting 3: the processes with 5 threads d) Setting 4: the processes with 9 threads (MRA is the Maximum Reachable Accuracy, each run indicates different random network topology parameters).

As we mentioned earlier, MRA can be calculated by analyzing the states of resources and the network topology in each run. In fact, it is possible that request requirements for a set of resources not be fully obtained in the system. This might happen when the requested set is not available (reserved), or never existed in the system. In such a case our approach tries to offer a similar set, which can fulfill most of the requirements. MRA specifies the border line to offer the most optimal set of resources for a specific process due to the realistic conditions of the resources in the system. The bar charts represent the average percentage of RAA with different settings and runs. As can be seen in the results, the RAA in all of the tests are almost close to the value of the maximum reachable accuracy which presents a resalable level of accuracy for the resource allocation in the proposed resource management approach. The result shows that, the RAA for small processes (i.e., the process with small number of threads) (Figure 5.19.a and Figure 5.19.b) in comparison to the larger processes (Figure 5.19.c and Figure 5.19.d) is closer to the maximum in overall.

Figure 5.20 shows the average inaccuracy ratio for different types of processes with various levels of dependency between their threads. It can be seen that the inaccuracy ratio grows for the larger processes with higher level of thread dependency. However the inaccuracy slope is still less than the dependency slope. The reason for this behavior is that larger processes, with higher dependencies, have more requirements for their required set of resources,

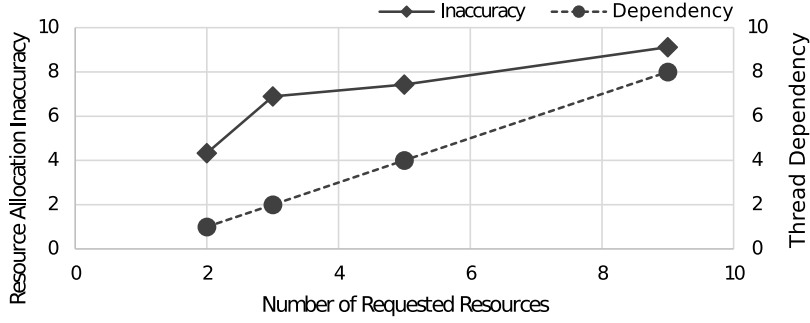


Figure 5.20: Resource Allocation Inaccuracy ratio for the processes with different number of threads and threads' dependency.

which can't be potentially fulfilled in the system in comparison to the smaller processes with lower dependencies. When the system is not able to provide the qualified set of resources considering all the process requirements, it returns the optimal results that partially meet the requirements. And this increases the inaccuracy ratio. Our approach tries to provide the most qualified set of resources for allocation. However if it is not feasible to obtain, (for reasons such as unavailability of the resources and system limitations) it offers the best possible set of resources.

5.5.3 Scalability and Performance

In the rest of this section, we evaluate the performance of our resource management model with respect to scalability. To do this we conduct a simulation scenario based on the simulation parameters mentioned in Table 5.6 with the following changes, presented in Table 5.7.

Table 5.7: Simulation Parameters for Scalability Evaluation

Parameter	Values
Requested Resources for each Process	20
Frequency of Target Resources (FTR)	1650
Physical network size	27500 cores
Process Duration by sec	Weibull($\lambda=3.58, k=2.40$)
Querying Interval by ms	Exponential($\beta=4000$)
Consecutive Query Runs per RMC	1000
Number of Active RMCs	275

We simulate a distributed dynamic computing environment containing 27500 computing resources, in which a constant number of RMCs (i.e., active RMCs) simultaneously handle the incoming resource requests from their PM clients. The time interval between each pair of consecutive queries issued by PMs (i.e., the arrival rate of incoming resource requests from PMs) is defined by Exponential distribution. We also assume that each active RMC only receives 1000 consecutive resource requests from PMs over the simulation time. RMCs assign and reserve the proper resources for each PM request. The resources reserved for each process be released after the execution time, which is defined by a Weibull distribution. We measure the system overhead caused by resource management components (i.e., number of transaction messages between RMCs, RRs and RPS to handle a resource request issued by a PM) as well as the RMC latency for requests over time.

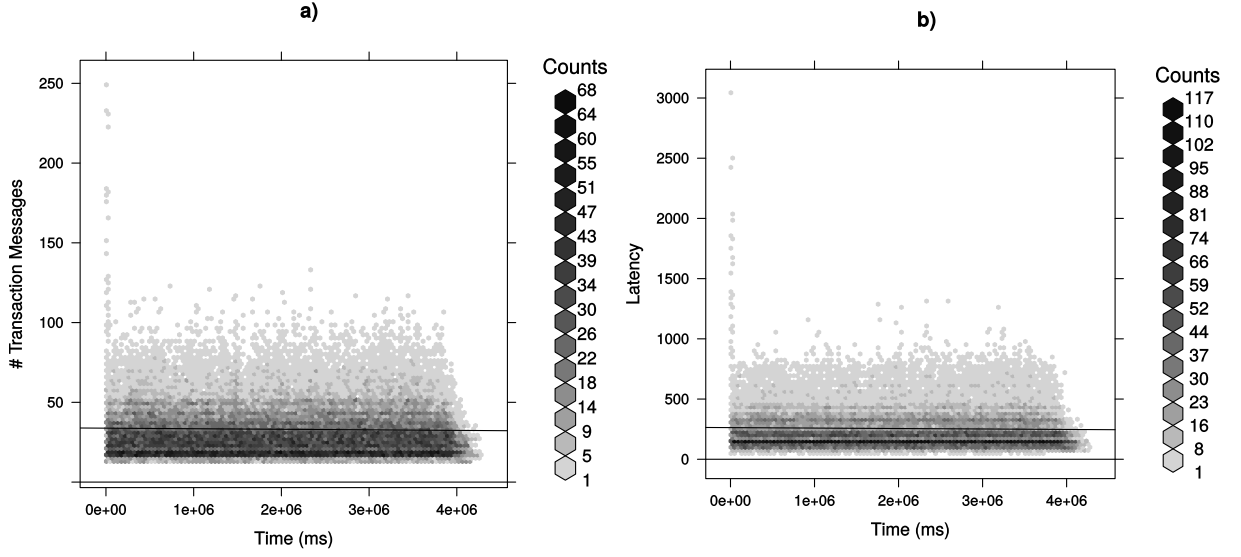


Figure 5.21: Hexbin plots for the number of transaction messages and the latency for each resource request over time.

Figure 5.21-a and Figure 5.21-b present the Hexbin plots for resource management overhead and latency over simulation time (by millisecond). Each data point in the graphs represents the result of a single PM resource request. The darker points in the graphs (i.e., the high density points) represent states that have a higher probability of occurrence in comparison to the lighter points. As we can see in these graphs, the majority of the queries results, particularly the high density data points, fall on or below the regression line. This illustrates that our resource management scheme is highly scalable over time and can efficiently maintain its performance under natural churn caused by high frequent resource reservations and resource releases in highly dynamic computing environment.

Figure 5.22-a and Figure 5.22-b illustrate that at least 80% of the resource requests are managed by their correspondent RMCs with less than 50 transaction messages which shows a reasonable overhead. But for less than 20% of the resource requests, the resource management overhead is a value between 50 to 250 messages. This happens mainly due to resource unavailability, rare resources and resource contentions. In fact, for larger process execution times, it takes longer for the reserved resources to be released by the original requesters and this leads to higher rate of unavailability for the occupied resources in the system. In this regard, the frequent resources in the system, might become rare resources over time with higher cost of discovery. In addition, the rate of resource unavailability and the speed with which the resources become rare will accelerated particularly when the overall task duration (i.e., process execution time) exceeds the overall querying interval. Figure 5.22-c and Figure 5.22-d also present the similar results for the average querying latency to process the requests that were initiated by different PM components in the system over simulation time.

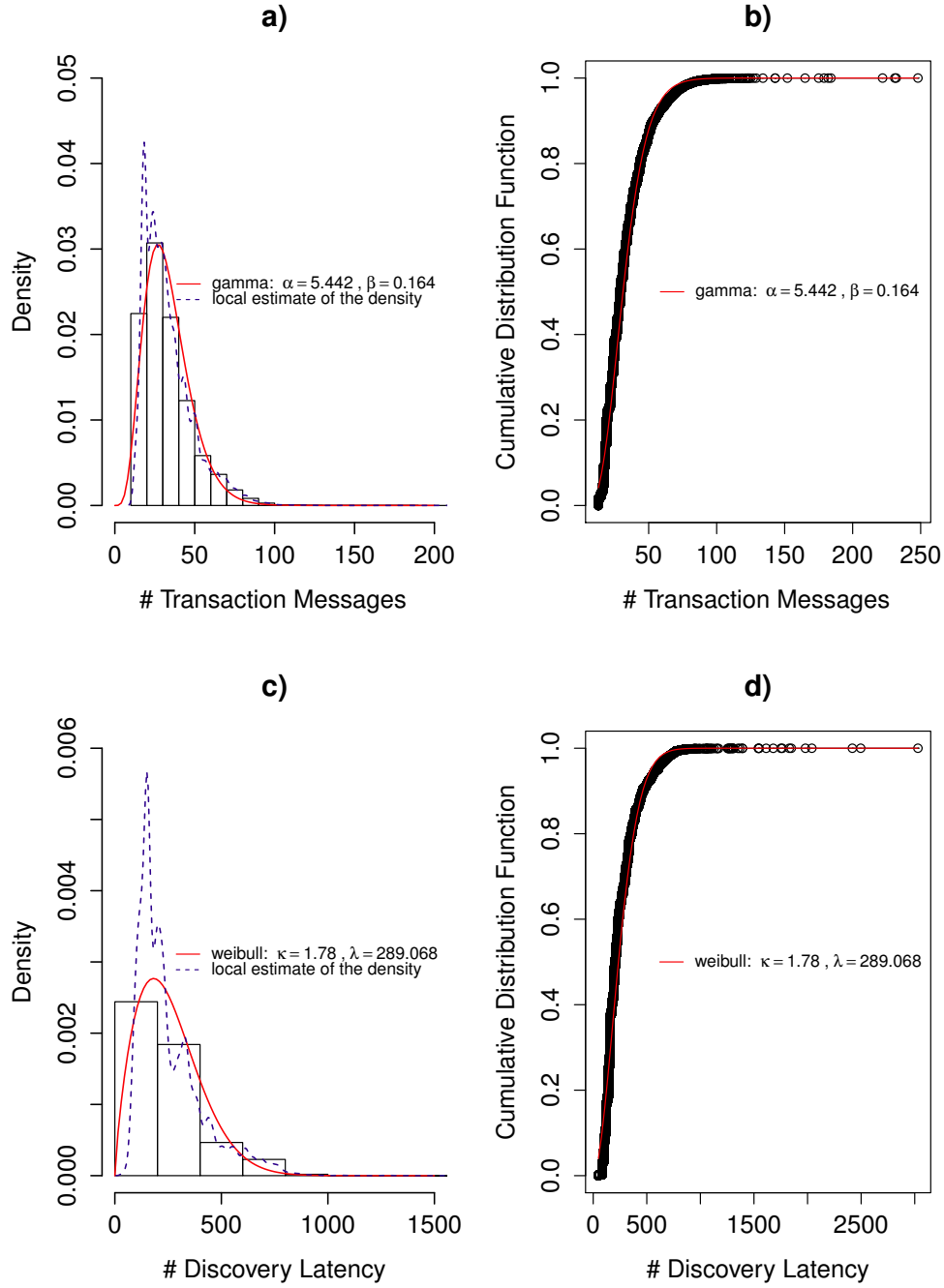


Figure 5.22: a and b) Histogram, density plot and CDF plot for the number of transaction messages between resource management components over simulation time, c and d) Histogram, density plot and CDF plot for the querying latency for the PM requesters over simulation time

Chapter 6

Concluding Remarks

In this chapter, we summarize the obtained results and provide some conclusive statements towards the key approaches investigated in this thesis, with some directions for future work.

6.1 Summary of the Thesis

In a large scale system, where we have a pool of distinct processors, the enabling technology for enhancing the whole throughput of the system is resource sharing. This means that for overloaded processors we can migrate the overloaded processes to other potential processors in the network. But resources should be found before the resource sharing, resource allocation and execution migration could be done. Resource discovery as a component of a distributed Operating System (OS) will be employed to discover an efficient set of available processing resources that could match the application requirements. The discovery latency has direct effect on the cost of migration and execution migration is not beneficial when resource discovery cannot provide information services in an acceptable time.

Unlike resource discovery for various other purposes and domains, resource discovery for a large many-core environment is very sensitive to the discovery performance and it could be useless when it cannot satisfy the minimal parametric conditions of the system environment. Scalability could be considered as one of the basic problems of resources discovery in future many-core systems which is a generic challenge for majority of the research works in this area. The scalability problems also refer to the methods for description of resources and the discovery procedure. These mechanisms must propose techniques and algorithms that are efficiently extendable for various number of resources. On the other hand the generated discovery overhead must be independent from network size. This is not fully attainable but we can make an effort to keep the discovery overhead almost constant with increasing the number of resources.

In **Chapter 2**, we investigated the resource discovery approaches, methods, and techniques in the current literature, in general and particularly with respect to the requirements of large-scale computing. This chapter also provides design guidelines for building resource discovery approaches for large scale distributed computing environments. In-order to build a resource discovery solution there are several fundamental open issues and involved elements, which have significant impact on the resource discovery's final behavior, operation, functionality, and even the way that it can deal with some challenging issues and objectives. This chapter discussed all the major common involved concepts, structures, techniques and functionalities

that shape the critical aspects of every resource discovery models. We categorized all these aspects in three groups consisting of structures, methods and issues. Accordingly, we classified the general steps and requirements to construct a novel resource discovery solution in three phases containing pre-discovery phase, discovery phase and post-discovery phase.

We then discussed the details of involved elements for each phase. Pre-discovery elements contain all the things that should be prepared and configured before discovery usage (e.g. stating a query). These elements are more relevant to the underlying computing environment, network topology data infrastructure, representation of resources (resource abstraction), and resource information distribution. Discovery elements include the current major possible relevant techniques, strategies and methods that can be used to guide queries between distributed entities in order to locate and discover resources. Search algorithms, mechanisms for packet propagation, querying strategies, resource information delivery, data synchronization between distributed resource provides, information summarization techniques are examples of such elements. And finally post discovery elements cover all the concepts and terms that express, demonstrate or evaluate the expected or desired achievements of resource discovery procedure. Depending on the specific environment, applications and objectives that a discovery protocol has been designed for, number of certain functionalities, features and performance factors have become critically important to attain. For example, for the resource discovery in wireless sensor networks, energy efficiency is one of the critical performance factor while in the area of large scale computing, scalability has the key role.

In **Chapter 3**, we proposed a new hierarchical resource description model, having this aspect of how the hardware resource information is distributed between computing-enabled peers (resources), and how the hardware resource capabilities can be described and derived from hardware descriptions. The proposed hierarchical hardware modeling and resource description mechanism is able to abstract the characteristics and behavior of the large scale many-core-enabled computing systems, in a way that both computational and communicational system properties are well represented to provide a close estimation of the real system, while avoiding to describe the hardware at the cycle-accurate level. The depth of hierarchy (i.e. number of layers in resource description model) and the definition of each layer also can range from very high level (e.g. super clusters, clusters) to very low level (e.g. processing core, Arithmetic Logic Units (ALUs)) depending on architectural designing aspects. The proposed resource description model is far too detailed and general for resource discovery and mapping, while dealing with aspects such as capturing static and dynamic capabilities of hardware resources, and making distribution of hardware information scalable.

In **Chapter 4**, we partition our work into three parts: self-organization, resource discovery and resource management. Section 4.2.1 imposed some assumptions on the design process, which can be considered as essential requirements for resource discovery in a fully decentralized and distributed fashion. Following that, in the first phase, we introduced a hierarchical proximity-aware self-deployment and self-configuration algorithm to alter the underlying network topology in a way that the optimal clustering of the nodes in different layers of hierarchy can be performed dynamically. The simulation results showed that the proposed grouping mechanism supports the load balance while it composes the groups in the layers with respect to the required locality and clustering requirements. Furthermore, the experiment results proved the scalability and efficiency of the proposed algorithm for different system sizes.

In the second part of this chapter, we proposed HARD, a novel efficient and highly scalable resource-discovery approach which deals with the resource discovery requirements in large

scale distributed computing environments such as high-hierarchy, high-heterogeneity and high-scalability and dynamicity. We used the algorithm, presented in the first part of this chapter for self-organization and self-adaptation of processing resources information in the system to organize the computing resources into distributed hierarchies according to the hierarchical resource description model (i.e. multi-layered resource description), proposed in chapter 3. We presented the details of HARD mechanisms in section 4.5. Following that, in section 4.6, several different algorithms and adapted approaches (such as distributed hash tables, distributed probability tables and any-casting) have been implemented at different layers in order to efficiently guide queries to proper resources within and across layers in the system.

In the third part, we proposed and developed a novel resource management scheme for future peta-scale many-core-enabled computing systems, based on HARD3, called ElCore, in section 4.7. The proposed architecture contains a set of modules which will dynamically be instantiated on the nodes in the distributed system on demand. Our approach provides flexibility to allocate the required set of resources for various types of processes/applications. It also can be considered as a generic solution (with respect to the general requirements of large scale computing environments) which brings a set of interesting features (such as auto-scaling, multitancy, multi-dimensional mapping, etc.,) to facilitate its easy adaptation to any distributed technology (such as SOA, Grid and HPC many-core). For example, ElCore can support auto-scale (the ability to provide budget to more-or-less expensive tasks) in two different aspects: dynamic resource allocation and dynamic module instantiation. System module instances would be automatically created when they are needed. And each module instance would be dependent on the type of application (e.g., real time, HPC, etc.), the computing resource (where the application starts to be executed), heterogeneity of resources, positioning of the resources in the system, network topology and interconnection between resources. Moreover, leveraging discovery components (RR-RPs) enables our resource management platform to dynamically find and allocate available resources that guarantee the QoS parameters on demand.

In **Chapter 5**, we presented comprehensive simulation results to evaluate HARD (the proposed resource discovery model). The achieved results assured the significant scalability and efficiency of HARD3 over highly heterogeneous, hierarchical and dynamic computing environments with respect to several scalability and efficiency aspects while supporting flexible and complex queries with guaranteed discovery results accuracy. We further showed that HARD outperforms different other potential strategies. We also presented simulation results to evaluate ElCore (the proposed resource management architecture). The simulation results proved that, using our approach, the mapping between processes and resources can be done with high level of accuracy which potentially leads to a significant enhancement in the overall system performance.

6.2 Future Research Directions

There are number of possibilities for future research. As we already mentioned in Chapter 4, the problem of resource identification and discovery, can be structured along the following two questions: first, how to identify the resources required for a given process? or how to identify the hardware resources requirements from either the static (source code) or the dynamic (runtime) behavior of the program (i.e., Resource Identification)? and second, how to find

the required resources for a given query (i.e., Resource Discovery)? In this thesis, we focused on the second question and presented a resource discovery model for large scale computing systems. But an investigation considering the first question (resource identification problem) can be performed in the future.

Until now, our proposed resource discovery and management approaches have been used for S(o)OS [4] which can be considered as an example of DOSs. The S(o)OS project developed an Operating System model geared towards future large scale, distributed infrastructures, considering the current trends in processor manufacturing including in particular the memory wall problem, many-core systems, the growing degree of heterogeneity and the different hierarchies within the infrastructure, in terms of communication and storage. S(o)OS has been evaluated through an architecture simulation, called S(o)OS Simulator (SoOSim) platform [468], which enables assessing the key performance characteristics of S(o)OS with respect to the scalability, heterogeneity, performance and timing constraints. The evaluation through SoOSim showed that by invoking our proposed resource discovery and management modules (HARD3 and ElCore components) in addition to other OS components, S(o)OS can exhibit considerably high degrees of scalability, heterogeneity, and performance for applications executions on very large scale distributed computing infrastructures. Furthermore, using ElCore, S(o)OS can incorporate scheduling policies that increase the resource utilization and thus can further improve performance. Since we developed HARD3 and ElCore according to service-oriented and component based design architecture, these components (and even any other instantiation of HARD) can easily be adapted and leveraged for future DOSs with modularity supports.

In this work, in order to demonstrate the functionalities and performance of our proposed resource discovery, we developed and implemented an instantiation of HARD (i.e. HARD3) which can be applied generally to large scale computing systems and particularly to future many-core systems. However, depending on the design objectives, computing environments, resource discovery constraints and requirements, different instantiation of HARD can be realized given different design variables (such as resource and layering definitions, within and across layers adapted mechanisms, etc.), purposes, and contexts of use for the target computing environments. For example, HARD, due to its adaptability and versatile nature, can be implemented for distributed resource discovery in different areas, computational and communication disciplines and on diverse hardware infrastructures such as wireless sensor networks, internet of things, software-defined networking and ubiquitous computing.

References

- [1] J. Maassen, N. Drost, H. E. Bal, and F. J. Seinstra, “Towards jungle computing with ibis/constellation,” in Proceedings of the 2011 Workshop on Dynamic Distributed Data-intensive Applications, Programming Abstractions, and Systems, 3DAPAS ’11, (New York, NY, USA), pp. 7–18, ACM, 2011.
- [2] H. Sutter, “The free lunch is over: A fundamental turn toward concurrency in software,” Dr. Dobbs’s Journal, vol. 30, no. 3, pp. 202–210, 2005.
- [3] F. J. Seinstra, J. Maassen, R. V. Van Nieuwpoort, N. Drost, T. Van Kessel, B. Van Werkhoven, J. Urbani, C. Jacobs, T. Kielmann, and H. E. Bal, “Jungle computing: Distributed supercomputing beyond clusters, grids, and clouds,” in Grids, Clouds and Virtualization, pp. 167–197, Springer, 2011.
- [4] L. Schubert and A. Kipp, “Principles of service oriented operating systems,” in Networks for Grid Applications (P. Vicat-Blanc Primet, T. Kudoh, and J. Mambretti, eds.), vol. 2 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 56–69, Springer Berlin Heidelberg, 2009.
- [5] The S[o]OS Consortium , “S(o)OS (Service-oriented Operating System): Resource-independent execution support on exa-scale systems.” <http://www.soos-project.eu/>, 2010-2013. [Online: accessed 9-January-2017].
- [6] A. Clematis, A. Corana, D. D’Agostino, A. Galizia, and A. Quarati, “Matching jobs with resources: an application-driven approach,” Scalable Computing: Practice and Experience, vol. 11, no. 2, pp. 109–120, 2001.
- [7] A. Tikotekar, G. Vallée, T. Naughton, H. Ong, C. Engelmann, and S. L. Scott, An Analysis of HPC Benchmarks in Virtual Machine Environments, pp. 63–71. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.
- [8] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, “COTSon: Infrastructure for full system simulation,” SIGOPS Oper. Syst. Rev., vol. 43, pp. 52–61, Jan. 2009.
- [9] R. Bedichek, “Simnow: Fast platform simulation purely in software,” in Hot Chips, vol. 16, pp. 48–60, 2004.
- [10] A. Varga et al., “The omnet++ discrete event simulation system,” in Proceedings of the European simulation multiconference (ESM’2001), vol. 9, p. 65, sn, 2001.

- [11] I. Baumgart, B. Heep, and S. Krause, "Oversim: A scalable and flexible overlay framework for simulation and real network applications," in Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on, pp. 87–88, IEEE, 2009.
- [12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in Grid Computing Environments Workshop, 2008. GCE'08, pp. 1–10, Ieee, 2008.
- [13] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid computing-where HPC meets grid and cloud computing," Future Gener. Comput. Syst., vol. 27, pp. 440–453, May 2011.
- [14] N. J. Navimipour, A. M. Rahmani, A. H. Navin, and M. Hosseinzadeh, "Resource discovery mechanisms in grid systems: A survey," Journal of Network and Computer Applications, vol. 41, pp. 389–410, 2014.
- [15] A. Hameurlain, D. Cokuslu, and K. Erciyes, "Resource discovery in grid systems: a survey," Int. J. Metadata Semant. Ontologies, vol. 5, pp. 251–263, July 2010.
- [16] Z. Xin-Xin, Z. Zhen-Wan, K. Peng, and S. Ren-Jie, "A survey of resource discovery in mobile peer-to-peer networks," in Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on, pp. 122–125, April 2015.
- [17] M. I. Hassan and A. Abdullah, "Semantic-based grid resource discovery systems a literature review and taxonomy," in 2010 International Symposium on Information Technology, vol. 3, pp. 1286–1296, June 2010.
- [18] F. Sharifkhani and M. R. Pakravan, "A review of new advances in resource discovery approaches in unstructured P2P networks," in Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on, pp. 828–833, Aug. 2013.
- [19] A. Arunachalam and O. Sornil, "An analysis of the overhead and energy consumption in flooding, random walk and gossip based resource discovery protocols in mp2p networks," in 2015 Fifth International Conference on Advanced Computing Communication Technologies, pp. 292–297, Feb. 2015.
- [20] D. Lazaro, J. M. Marques, J. Jorba, and X. Vilajosana, "Decentralized resource discovery mechanisms for distributed computing in peer-to-peer environments," ACM Comput. Surv., vol. 45, pp. 54:1–54:40, Aug. 2013.
- [21] K. Vanthournout, G. Deconinck, and R. Belmans, "A taxonomy for resource discovery," Personal Ubiquitous Comput., vol. 9, pp. 81–89, Mar. 2005.
- [22] S. Davtyan, Resource Discovery and Cooperation in Decentralized Systems. PhD thesis, University of Connecticut, University of Connecticut, Storrs, Connecticut, United States, Sept. 2014. Doctoral Dissertations.Paper 425.
- [23] N. Antonopoulos, Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications: Models, Methodologies and Applications. IGI Global, 2010.

- [24] K. Krauter, R. Buyya, and M. Maheswaran, “A taxonomy and survey of grid resource management systems for distributed computing,” Software: Practice and Experience, vol. 32, pp. 135–164, Feb. 2002.
- [25] D. Talia and P. Trunfio, “Peer-to-peer protocols and grid services for resource discovery on grids,” in Grid Computing The New Frontier of High Performance Computing (L. Grandinetti, ed.), vol. 14 of Advances in Parallel Computing, pp. 83–103, North-Holland, 2005.
- [26] D. C. Erdil, Adaptive Dissemination Protocols for Hybrid Grid Resource Scheduling. PhD thesis, State University of New York at Binghamton, Binghamton, NY, USA, 2007. AAI3289113.
- [27] A. K. Shaikh, S. M. Alhashmi, and R. Parthiban, “A semantic-based centralized resource discovery model for grid computing,” in Grid and Distributed Computing, pp. 161–170, Springer, 2011.
- [28] I. Foster and C. Kesselman, “Globus: A metacomputing infrastructure toolkit,” International Journal of High Performance Computing Applications, vol. 11, no. 2, pp. 115–128, 1997.
- [29] C. Germain, V. Neri, G. Fedak, and F. Cappello, “Xtremweb: building an experimental platform for global computing,” in Grid Computing—GRID 2000, pp. 91–101, Springer, 2000.
- [30] A. Chien, B. Calder, S. Elbert, and K. Bhatia, “Entropia: architecture and performance of an enterprise desktop grid system,” Journal of Parallel and Distributed Computing, vol. 63, no. 5, pp. 597–610, 2003.
- [31] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, et al., “Adaptive computing on the grid using apples,” Parallel and Distributed Systems, IEEE Transactions on, vol. 14, no. 4, pp. 369–382, 2003.
- [32] A. K. Shaikh, S. M. Alhashmi, and R. Parthiban, “A semantic decentralized chord-based resource discovery model for grid computing,” in Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on, pp. 142–148, Dec. 2011.
- [33] X. Wang and L. F. Kong, “Resource clustering based decentralized resource discovery scheme in computing grid,” in Machine Learning and Cybernetics, 2007 International Conference on, vol. 7, pp. 3859–3863, Aug 2007.
- [34] R.-S. Chang and M.-S. Hu, “A resource discovery tree using bitmap for grids,” Future Gener. Comput. Syst., vol. 26, pp. 29–37, Jan. 2010.
- [35] L. M. Khanli and S. Kargar, “Frtd: footprint resource discovery tree for grids,” Future Generation Computer Systems, vol. 27, no. 2, pp. 148–156, 2011.
- [36] A. Forestiero, C. Mastroianni, and G. Spezzano, “A decentralized ant-inspired approach for resource management and discovery in grids,” Multiagent and Grid Systems, vol. 3, no. 1, pp. 43–63, 2007.

- [37] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on, pp. 181–194, IEEE, 2001.
- [38] G. Alliance, "Gt information services: Monitoring & discovery system (mds)," 2005.
- [39] D. Abramson, J. Giddy, and L. Kotler, "High performance parametric modeling with nimrod/g: Killer application for the global grid?," in ipdps, p. 520, IEEE, 2000.
- [40] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," Cluster Computing, vol. 5, no. 3, pp. 237–246, 2002.
- [41] R. Byrom, B. Coghlan, A. Cooke, R. Cordenonsi, L. Cornwall, M. Craig, A. Djaoui, A. Duncan, S. Fisher, A. Gray, S. Hicks, S. Kenny, J. Leake, O. Lyttleton, J. Magowan, R. Middleton, W. Nutt, D. O'Callaghan, N. Podhorszki, P. Taylor, J. Walk, and A. Wilson, "Fault tolerance in the r-GMa information and monitoring system," in Proceedings of the 2005 European Conference on Advances in Grid Computing, EGC'05, (Berlin, Heidelberg), pp. 751–760, Springer-Verlag, 2005.
- [42] R. Byrom, B. Coghlan, A. Cooke, R. Cordenonsi, L. Cornwall, A. Datta, A. Djaoui, L. Field, S. Fisher, S. Hicks, S. Kenny, J. Magowan, W. Nutt, D. O'Callaghan, M. Oevers, N. Podhorszki, J. Ryan, M. Soni, P. Taylor, A. Wilson, and X. Zhu, "The canonical-producer: An instrument monitoring component of the relational grid monitoring architecture (r-GMa)," in Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, ISPDC '04, (Washington, DC, USA), pp. 232–237, IEEE Computer Society, 2004.
- [43] X. Zhang, J. L. Freschl, and J. M. Schopf, "Scalability analysis of three monitoring and information systems: Mds2, r-GMa, and hawkeye," Journal of Parallel and Distributed Computing, vol. 67, pp. 883–902, Aug. 2007.
- [44] Z. Pan, A. Qasem, S. Kanitkar, F. Prabhakar, and J. Heflin, "Hawkeye: A practical large scale demonstration of semantic web integration," in Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part II, OTM'07, (Berlin, Heidelberg), pp. 1115–1124, Springer-Verlag, 2007.
- [45] J. Yu, S. Venugopal, and R. Buyya, "A market-oriented grid directory service for publication and discovery of grid service providers and their services," J. Supercomput., vol. 36, pp. 17–31, Apr. 2006.
- [46] J. Yu, S. Venugopal, and R. Buyya, "A market-oriented grid directory service for publication and discovery of grid service providers and their services," J. Supercomput., vol. 36, pp. 17–31, Apr. 2006.
- [47] X. Zhang and J. Schopf, "Performance analysis of the globus toolkit monitoring and discovery service, mds2," in the 2004 Ieee International Performance, Computing, and Communications Conference, pp. 843–849, 2004.

- [48] H. N. L. C. Keung and S. A. Jarvis, “Performance modelling of a self-adaptive and self-optimising resource monitoring system for dynamic grid environments,” 2004.
- [49] I. Diaz, G. Fernandez, M. J. Martinm, P. Gonzalez, and J. Tourino, “Integrating the common information model with mds4,” in Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing, GRID ’08, (Washington, DC, USA), pp. 298–303, IEEE Computer Society, 2008.
- [50] J. M. Schopf, L. Pearlman, N. Miller, C. Kesselman, I. Foster, M. D’Arcy, and A. Chervenak, “Monitoring the grid with the globus toolkit MDS4,” in Journal of Physics: Conference Series, vol. 46, p. 521, IOP Publishing, IOP Publishing, 2006.
- [51] M. L. Massie, B. N. Chun, and D. E. Culler, “The ganglia distributed monitoring system: Design, implementation and experience,” Parallel Computing, vol. 30, p. 2004, 2003.
- [52] H. Sun, J. Huai, Y. Liu, and R. Buyya, “RCt: A distributed tree for supporting efficient range and multi-attribute queries in grid computing,” Future Generation Computer Systems, vol. 24, no. 7, pp. 631–643, 2008.
- [53] A. Naseer and L. K. Stergioulas, “Resource discovery in grids and other distributed environments: States of the art,” Multiagent and Grid Systems, vol. 2, no. 2, pp. 163–182, 2006.
- [54] I. Foster and A. Iamnitchi, “On death, taxes, and the convergence of peer-to-peer and grid computing,” in In 2nd International Workshop on Peer-to-Peer Systems (IPTPS’03), pp. 118–128, 2003.
- [55] D. Talia and P. Trunfio, “Toward a synergy between P2P and grids,” Internet Computing, IEEE, vol. 7, no. 4, pp. 96–95, 2003.
- [56] N. Hidalgo, L. Arantes, P. Sens, and X. Bonnaire, “Echo: Efficient complex query over {DHT} overlays,” Journal of Parallel and Distributed Computing, vol. 88, pp. 31 – 45, 2016.
- [57] J. Albrecht et al., “Design and implementation trade-offs for wide-area resource discovery,” Acm Transactions on Internet Technology, vol. 8(4), 2008.
- [58] C. Mastroianni, D. Talia, and O. Verta, “A super-peer model for resource discovery services in large-scale grids,” Future Generation Computer Systems, vol. 21, no. 8, pp. 1235–1248, 2005.
- [59] A. R. Butt, R. Zhang, and Y. C. Hu, “A self-organizing flock of condors,” in Proceedings of the 2003 ACM/IEEE conference on Supercomputing, SC ’03, (New York, NY, USA), pp. 42–, ACM, 2003.
- [60] J. A. Torkestani, “A distributed resource discovery algorithm for {P2P} grids,” Journal of Network and Computer Applications, vol. 35, no. 6, pp. 2028–2036, 2012.
- [61] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi, “Peer-to-peer resource discovery in grids: Models and systems,” Future Generation Computer Systems, vol. 23, no. 7, pp. 864–878, 2007.

- [62] K. Lee, T. Choi, P. O. Boykin, and R. J. Figueiredo, "Matchtree: Flexible, scalable, and fault-tolerant wide-area resource discovery with distributed matchmaking and aggregation," Future Gener. Comput. Syst., vol. 29, pp. 1596–1610, Aug. 2013.
- [63] G. Pipan, "Use of the {TRIPOD} overlay network for resource discovery," Future Generation Computer Systems, vol. 26, no. 8, pp. 1257–1270, 2010.
- [64] T. Ghafarian, H. Deldari, B. Javadi, M. H. Yaghmaee, and R. Buyya, "Cycloidgrid: A proximity-aware P2P-based resource discovery architecture in volunteer computing systems," Future Gener. Comput. Syst., vol. 29, pp. 1583–1595, Aug. 2013.
- [65] H. Shen, "A p2p-based intelligent resource discovery mechanism in internet-based distributed systems," Journal of Parallel and Distributed Computing, vol. 69, no. 2, pp. 197 – 209, 2009.
- [66] T. Welch, "A technique for high-performance data compression," Computer, vol. 17, no. 6, pp. 8–19, 1984.
- [67] A. R. Butt, R. Zhang, and Y. C. Hu, "A self-organizing flock of condors," Journal of Parallel and Distributed Computin, vol. 66, no. 1, pp. 145 – 161, 2006.
- [68] M. Xu, S. Zhou, and J. Guan, "A new and effective hierarchical overlay structure for peer-to-peer networks," Comput. Commun., vol. 34, pp. 862–874, May 2011.
- [69] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "Skipnet: a scalable overlay network with practical locality properties," in Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4, USITS'03, (Berkeley, CA, USA), pp. 9–9, USENIX Association, 2003.
- [70] J. Aspnes and G. Shah, "Skip graphs," ACM Trans. Algorithms, vol. 3, p. 37, Nov. 2007.
- [71] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Trans. Netw., vol. 11, pp. 17–32, Feb. 2003.
- [72] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," SIGCOMM Comput. Commun. Rev., vol. 31, pp. 161–172, Aug. 2001.
- [73] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Middleware '01, (London, UK, UK), pp. 329–350, Springer-Verlag, 2001.
- [74] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," IEEE J.Sel. A. Commun., vol. 22, pp. 41–53, Sept. 2006.
- [75] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Puceva, and R. Schmidt, "P-grid: a self-organizing structured P2P system," SIGMOD Rec., vol. 32, pp. 29–33, Sept. 2003.

- [76] V. March, Y. M. Teo, and X. Wang, “Dgrid: a DHT-based resource indexing and discovery scheme for computational grids,” in Proceedings of the fifth Australasian symposium on ACSW frontiers - Volume 68, ACSW '07, (Darlinghurst, Australia, Australia), pp. 41–48, Australian Computer Society, Inc., 2007.
- [77] A. R. Bharambe, M. Agrawal, and S. Seshan, “Mercury: Supporting scalable multi-attribute range queries,” in Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '04, (New York, NY, USA), pp. 353–366, ACM, 2004.
- [78] C. Tang, Z. Xu, and M. Mahalingam, “psearch: information retrieval in structured overlays,” SIGCOMM Comput. Commun. Rev., vol. 33, pp. 89–94, Jan. 2003.
- [79] A. K. Shaikh, S. M. Alhashmi, and R. Parthiban, “A semantic decentralized chord-based resource discovery model for grid computing,” in Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems, ICPADS '11, (Washington, DC, USA), pp. 142–148, IEEE Computer Society, 2011.
- [80] Z. Chen, L. Wu, J. Zhang, X. Hu, and Y. Xu, “Heuristic resource discovery in p2p network,” in Proceedings of the 25th international conference on Industrial Engineering and Other Applications of Applied Intelligent Systems: advanced research in applied artificial intelligence, IEA/AIE'12, (Berlin, Heidelberg), pp. 333–342, Springer-Verlag, 2012.
- [81] S. Sotiriadis, N. Bessis, and N. Antonopoulos, “Using self-led critical friend topology based on P2P chord algorithm for node localization within cloud communities,” in Proceedings of the 2011 International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS '11, (Washington, DC, USA), pp. 490–495, IEEE Computer Society, 2011.
- [82] A. Forestiero, C. Mastroianni, and M. Meo, “Self-chord: A bio-inspired algorithm for structured P2P systems,” in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, (Washington, DC, USA), pp. 44–51, IEEE Computer Society, 2009.
- [83] X. Ke, S. Meina, and S. Junde, “An improved P2P lookup protocol model,” Cluster Computing, vol. 13, pp. 199–211, June 2010.
- [84] S. Kniesburges, A. Koutsopoulos, and C. Scheideler, “Re-chord: a self-stabilizing chord overlay network,” in Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures, SPAA '11, (New York, NY, USA), pp. 235–244, ACM, 2011.
- [85] E. J.-L. Lu, Y.-F. Huang, and S.-C. Lu, “ML-chord: A multi-layered P2P resource sharing model,” J. Netw. Comput. Appl., vol. 32, pp. 578–588, May 2009.
- [86] Y.-J. Joung and J.-C. Wang, “Chord2: A two-layer chord for reducing maintenance overhead via heterogeneity,” Comput. Netw., vol. 51, pp. 712–731, Feb. 2007.
- [87] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, “Tapestry: An infrastructure for fault-tolerant wide-area location,” tech. rep., University of California at Berkeley, Berkeley, CA, USA, 2001.

- [88] D. Battre, A. Hoing, M. Raack, U. Rerrer-Brusch, and O. Kao, “Extending pastry by an alphanumerical overlay,” in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID ’09, (Washington, DC, USA), pp. 36–43, IEEE Computer Society, 2009.
- [89] Z. Guo, L. Min, S. Yang, and H. Yang, “An enhanced p4p-based pastry routing algorithm for P2P network,” in Proceedings of the 2010 IEEE International Conference on Granular Computing, GRC ’10, (Washington, DC, USA), pp. 687–691, IEEE Computer Society, 2010.
- [90] T. Wang and R.-H. Di, “A semantic web service discovery model based on pastry system,” in Proceedings of the Fifth Annual ChinaGrid Conference, CHINAGRID ’10, (Washington, DC, USA), pp. 205–208, IEEE Computer Society, 2010.
- [91] J. Chen and L. Wang, “Improved hierarchical network model based on pastry scheme,” in Proceedings of the 2009 Second International Workshop on Knowledge Discovery and Data Mining, WKDD ’09, (Washington, DC, USA), pp. 859–862, IEEE Computer Society, 2009.
- [92] D. Spence and T. Harris, “Xenosearch: Distributed resource discovery in the xenoserver open platform,” in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, HPDC ’03, (Washington, DC, USA), pp. 216–, IEEE Computer Society, 2003.
- [93] I. Chang-Yen, D. Smith, and N.-F. Tzeng, “Structured peer-to-peer resource discovery for computational grids,” in Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities, MG ’08, (New York, NY, USA), pp. 6:1–6:8, ACM, 2008.
- [94] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram, “P-ring: an efficient and robust P2P range index structure,” in Proceedings of the 2007 ACM SIGMOD international conference on Management of data, SIGMOD ’07, (New York, NY, USA), pp. 223–234, ACM, 2007.
- [95] A. N. Crainiceanu, Answering complex queries in peer-to-peer systems. PhD thesis, Cornell University, Ithaca, NY, USA, 2006. AAI3227233.
- [96] P. Ganesan, M. Bawa, and H. Garcia-Molina, “Online balancing of range-partitioned data with applications to peer-to-peer systems,” in Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, VLDB ’04, pp. 444–455, VLDB Endowment, 2004.
- [97] G. Pirró, P. Trunfio, D. Talia, P. Missier, and C. Goble, “Ergot: A semantic-based system for service discovery in distributed infrastructures,” in Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pp. 263–272, 2010.

- [98] A. Crespo and H. Garcia-Molina, "Semantic overlay networks for p2p systems," in Proceedings of the Third international conference on Agents and Peer-to-Peer Computing, AP2PC'04, (Berlin, Heidelberg), pp. 1–13, Springer-Verlag, 2005.
- [99] Y. Tao, H. Jin, S. Wu, and X. Shi, "Scalable dht- and ontology-based information service for large-scale grids," Future Generation Computer Systems, vol. 26, no. 5, pp. 729 – 739, 2010.
- [100] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks," Comput. Netw., vol. 52, pp. 2097–2128, Aug. 2008.
- [101] A. Montresor, "A robust protocol for building superpeer overlay topologies," in Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on, pp. 202–209, 2004.
- [102] B. Beverly Yang and H. Garcia-Molina, "Designing a super-peer network," in Data Engineering, 2003. Proceedings. 19th International Conference on, pp. 49–60, 2003.
- [103] J. Li, "Grid resource discovery based on semantically linked virtual organizations," Future Generation Computer Systems, vol. 26, no. 3, pp. 361–373, 2010.
- [104] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks," Computer Networks, vol. 52, no. 11, pp. 2097–2128, 2008.
- [105] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Pouyoul, and B. Yeager, "Project jxta 2.0 super-peer virtual network," 2003.
- [106] D. Talia and P. Trunfio, "Peer-to-peer protocols and grid services for resource discovery on grids," Advances in Parallel Computing, vol. 14, pp. 83–103, 2005.
- [107] K. Truelove and A. Chasin, "Morpheus out of the underworld," The O’Rielly Network, 2001.
- [108] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on, pp. 23–32, 2002.
- [109] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in Peer-to-Peer Computing, 2001. Proceedings. First International Conference on, pp. 99–100, aug 2001.
- [110] S. Meng, C. Shi, D. Han, X. Zhu, and Y. Yu, A Statistical Study of Today’s Gnutella, pp. 189–200. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.
- [111] D. Talia and P. Trunfio, "Adapting a pure decentralized peer-to-peer protocol for grid services invocation," Parallel Processing Letters, vol. 15, no. 01n02, pp. 67–84, 2005.
- [112] S. H. Kwok and K. Y. Chan, "An enhanced gnutella P2P protocol: A search perspective," in Proceedings of the 18th International Conference on Advanced Information Networking and Applications - Volume 2, AINA '04, (Washington, DC, USA), pp. 599–, IEEE Computer Society, 2004.

- [113] A. C. Caminero, A. Robles-Gómez, S. Ros, R. Hernández, and L. Tobarra, “P2P-based resource discovery in dynamic grids allowing multi-attribute and range queries,” Parallel Computing, vol. 39, no. 10, pp. 615–637, 2013.
- [114] R. Brunner, A. C. Caminero, O. F. Rana, F. Freitag, and L. Navarro, “Network-aware summarisation for resource discovery in P2P-content networks,” Future Generation Computer Systems, vol. 28, no. 3, pp. 563–572, 2012.
- [115] J. Lee, P. Keleher, and A. Sussman, “Decentralized multi-attribute range search for resource discovery and load balancing,” The Journal of Supercomputing, vol. 68, no. 2, pp. 890–913, 2014.
- [116] I. Foster and C. Kesselman, The Grid 2: Blueprint for a new computing infrastructure. Elsevier, 2003.
- [117] M. Harchol-Balter, T. Leighton, and D. Lewin, “Resource discovery in distributed networks,” in Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC ’99, (New York, NY, USA), pp. 229–237, ACM, 1999.
- [118] D. A. Reed, “Grids, the teragrid, and beyond,” Computer, vol. 36, pp. 62–68, Jan. 2003.
- [119] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, “Condor-g: A computation management agent for multi-institutional grids,” Cluster Computing, vol. 5, pp. 237–246, July 2002.
- [120] G. Sabin, V. Sahasrabudhe, and P. Sadayappan, “Assessment and enhancement of meta-schedulers for multi-site job sharing,” in High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on, pp. 144–153, 2005.
- [121] R. Raman, M. Livny, and M. Solomon, “Matchmaking: Distributed resource management for high throughput computing,” in High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on, pp. 140–146, IEEE, 1998.
- [122] M. Litzkow, M. Livny, and M. Mutka, “Condor-a hunter of idle workstations,” in Distributed Computing Systems, 1988., 8th International Conference on, pp. 104–111, Jun 1988.
- [123] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, “Condor: A distributed job scheduler,” in Beowulf Cluster Computing with Linux, pp. 307–350, Cambridge, MA, USA: MIT Press, 2002.
- [124] D. Anderson, “Boinc: a system for public-resource computing and storage,” in Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on, pp. 4–10, 2004.
- [125] I. Foster, “Globus toolkit version 4: Software for service-oriented systems,” in Proceedings of the 2005 IFIP International Conference on Network and Parallel Computing, NPC’05, (Berlin, Heidelberg), pp. 2–13, Springer-Verlag, 2005.
- [126] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, “Falkon: A fast and light-weight task execution framework,” in Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC ’07, (New York, NY, USA), pp. 43:1–43:12, ACM, 2007.

- [127] S. Koussih, A. Acharya, and S. Setia, “Dodo: a user-level system for exploiting idle memory in workstation clusters,” in High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on, pp. 301–308, 1999.
- [128] G. Staples, “Torque resource manager,” in Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC ’06, (New York, NY, USA), ACM, 2006.
- [129] A. Chien, B. Calder, S. Elbert, and K. Bhatia, “Entropia: Architecture and performance of an enterprise desktop grid system,” J. Parallel Distrib. Comput., vol. 63, pp. 597–610, May 2003.
- [130] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, “Seti@home: An experiment in public-resource computing,” Commun. ACM, vol. 45, pp. 56–61, Nov. 2002.
- [131] V. S. Pande, I. Baker, J. Chapman, S. P. Elmer, S. Khaliq, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, et al., “Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing,” Biopolymers, vol. 68, no. 1, pp. 91–109, 2003.
- [132] S. Koussih, A. Acharya, and S. Setia, “Dodo: A user-level system for exploiting idle memory in workstation clusters,” in High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on, pp. 301–308, IEEE, 1999.
- [133] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility,” Future Gener. Comput. Syst., vol. 25, pp. 599–616, June 2009.
- [134] S. Di, C.-L. Wang, and L. Chen, “Ex-post efficient resource allocation for self-organizing cloud,” Comput. Electr. Eng., vol. 39, pp. 2342–2356, Oct. 2013.
- [135] D. C. Erdil, “Dependable autonomic cloud computing with information proxies,” in Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, pp. 1518–1524, IEEE, 2011.
- [136] W. Gentzsch, “Porting hpc applications to grids and clouds,” Cloud, Grid and High Performance Computing: Emerging Applications: Emerging Applications, p. 10, 2011.
- [137] F. Eljorje and J. Lee, “Cloudswitch: A state-aware monitoring strategy towards energy-efficient and performance-aware cloud data centers,” KSII Transactions on Internet & Information Systems, vol. 9, no. 12, 2015.
- [138] B. Di Martino, G. Cretella, and A. Esposito, “Cross-platform cloud apis,” in Cloud Portability and Interoperability, pp. 45–57, Springer, 2015.
- [139] A. Brogi, A. Ibrahim, J. Soldani, J. Carrasco, J. Cubo, E. Pimentel, and F. D’Andria, “SeacLOUDS: a european project on seamless management of multi-cloud applications,” ACM SIGSOFT Software Engineering Notes, vol. 39, no. 1, pp. 1–4, 2014.
- [140] A. Ionescu, “Standard interfaces for open source infrastructure as a service platforms,” Informatica Economica, vol. 19, no. 4, p. 68, 2015.

- [141] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CC-GRID '09, (Washington, DC, USA), pp. 124–131, IEEE Computer Society, 2009.
- [142] Y. Chen, X. Li, and F. Chen, "Overview and analysis of cloud computing research and application," in E-Business and E-Government (ICEE), 2011 International Conference on, pp. 1–4, IEEE, 2011.
- [143] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," Internet computing, IEEE, vol. 13, no. 5, pp. 14–22, 2009.
- [144] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (iaas) in cloud computing: A survey," Journal of Network and Computer Applications, no. 0, pp. –, 2013.
- [145] D. Milojevic, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," IEEE Internet Computing, vol. 15, pp. 11–14, Mar. 2011.
- [146] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, W. Emmerich, and F. Galán, "The reservoir model and architecture for open federated cloud computing," 2009.
- [147] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Muñoz, and G. Tofetti, "Reservoir - when one cloud is not enough," Computer, vol. 44, no. 3, pp. 44–51, 2011.
- [148] P. Wright, Y. L. Sun, T. Harmer, A. Keenan, A. Stewart, and R. Perrott, "A constraints-based resource discovery model for multi-provider cloud environments," Journal of Cloud Computing, vol. 1, no. 1, pp. 1–14, 2012.
- [149] M. Siddiqui and T. Fahringer, Grid Resource Management: On-demand Provisioning, Advance Reservation, and Capacity Planning of Grid Resources, ch. Semantics in the Grid: Towards Ontology-Based Resource Provisioning, pp. 157–177. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.
- [150] E. Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [151] W. C. W. Recommendation, "Owl web ontology language." <https://www.w3.org/TR/owl-features/>, 2004. [Online: accessed 9-January-2017].
- [152] C. Campo, M. Munoz, J. C. Perea, A. Marin, and C. Garcia-Rubio, "PDP and gsdl: A new service discovery middleware to support spontaneous interactions in pervasive systems," in Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOMW '05, (Washington, DC, USA), pp. 178–182, IEEE Computer Society, 2005.

- [153] S. Banerjee, S. Basu, S. Garg, S. Garg, S.-J. Lee, P. Mullan, and P. Sharma, “Scalable grid service discovery based on uddi,” in Proceedings of the 3rd International Workshop on Middleware for Grid Computing, MGC '05, (New York, NY, USA), pp. 1–6, ACM, 2005.
- [154] A. Zhu and Y. Xie, “Publishing and discovering knowledge services for design on uddi,” Int. J. Comput. Appl. Technol., vol. 23, pp. 31–40, Mar. 2005.
- [155] S. Pastore, “The service discovery methods issue: A web services uddi specification framework integrated in a grid environment,” J. Netw. Comput. Appl., vol. 31, pp. 93–107, Apr. 2008.
- [156] K. Tutschku, V. A. Mehri, A. Carlsson, K. V. Chivukula, and J. Christenson, “On resource description capabilities of on-board tools for resource management in cloud networking and NFv infrastructures,” in 2016 IEEE International Conference on Communications Workshops (ICC), pp. 442–447, May 2016.
- [157] V. Vaikuntanathan and P. Voulgaris, “Attribute based encryption using lattices,” June 2 2016. US Patent 20,160,156,465.
- [158] S. Fugkeaw and H. Sato, “Design and implementation of collaborative ciphertext-policy attribute-role based encryption for data access control in cloud,” Journal of Information Security Research, vol. 6, Sept. 2015.
- [159] G. Baranwal and D. P. Vidyarthi, “A fair multi-attribute combinatorial double auction model for resource allocation in cloud computing,” Journal of Systems and Software, vol. 108, pp. 60–76, 2015.
- [160] X. Yao, Z. Chen, and Y. Tian, “A lightweight attribute-based encryption scheme for the internet of things,” Future Generation Computer Systems, vol. 49, pp. 104–112, 2015.
- [161] N. S. Kumar, G. R. Lakshmi, and B. Balamurugan, “Enhanced attribute based encryption for cloud computing,” Procedia Computer Science, vol. 46, pp. 689–696, 2015.
- [162] H. Deng, Q. Wu, B. Qin, J. Domingo-Ferrer, L. Zhang, J. Liu, and W. Shi, “Ciphertext-policy hierarchical attribute-based encryption with short ciphertexts,” Information Sciences, vol. 275, pp. 370–384, 2014.
- [163] I. Sander and A. Jantsch, “System modeling and transformational design refinement in forsyde [formal system design],” Trans. Comp.-Aided Des. Integ. Cir. Sys., vol. 23, pp. 17–32, Nov. 2006.
- [164] A. Gill, T. Bull, A. Farmer, G. Kimmell, and E. Komp, “Types and type families for hardware simulation and synthesis: The internals and externals of kansas lava,” in Proceedings of the 11th International Conference on Trends in Functional Programming, TFP'10, (Berlin, Heidelberg), pp. 118–133, Springer-Verlag, 2011.
- [165] C. Baaij, M. Kooijman, J. Kuper, A. Boeijink, and M. Gerards, “Clash: Structural descriptions of synchronous hardware using haskell,” in Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on, pp. 714–721, 2010.

- [166] J. Kuper, C. Baaij, M. Kooijman, and M. Gerards, "Exercises in architecture specification using c #x03Bb;ash," in Specification Design Languages (FDL 2010), 2010 Forum on, pp. 1–6, 2010.
- [167] G. Klyne, J. J. Carroll, and B. McBride, "Resource description framework (rdf): Concepts and abstract syntax," W3C recommendation, vol. 10, 2004.
- [168] P. Ebenhoch, "Legal knowledge representation using the resource description framework (rdf)," in Proceedings of the 12th International Workshop on Database and Expert Systems Applications, DEXA '01, (Washington, DC, USA), pp. 369–, IEEE Computer Society, 2001.
- [169] K. S. Candan, H. Liu, and R. Suvarna, "Resource description framework: Metadata and its applications," SIGKDD Explor. Newsl., vol. 3, pp. 6–19, July 2001.
- [170] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. C. A. Klein, "Oil in a nutshell," in Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management, EKAW '00, (London, UK, UK), pp. 1–16, Springer-Verlag, 2000.
- [171] I. Horrocks, "Daml+oil: A reason-able web ontology language," in Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '02, (London, UK, UK), pp. 2–13, Springer-Verlag, 2002.
- [172] D. L. McGuinness, R. Fikes, J. Hendler, and L. A. Stein, "Daml+oil: An ontology language for the semantic web," IEEE Intelligent Systems, vol. 17, pp. 72–80, Sept. 2002.
- [173] L. Bangyong, T. Jie, L. Juanzi, and W. Kehong, "Using daml+oil to enhance search semantic," in Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence, WI '04, (Washington, DC, USA), pp. 465–468, IEEE Computer Society, 2004.
- [174] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen, "Reviewing the design of daml+oil: An ontology language for the semantic web," in Eighteenth National Conference on Artificial Intelligence, (Menlo Park, CA, USA), pp. 792–797, American Association for Artificial Intelligence, 2002.
- [175] M. H. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. V. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. P. Sycara, "Daml-s: Web service description for the semantic web," in Proceedings of the First International Semantic Web Conference on The Semantic Web, ISWC '02, (London, UK, UK), pp. 348–363, Springer-Verlag, 2002.
- [176] M. Montebello and C. Abela, "Daml enabled web services and agents in the semantic web," in Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, (London, UK, UK), pp. 46–58, Springer-Verlag, 2003.
- [177] S. A. Ludwig and S. M. S. Reyhani, "Introduction of semantic matchmaking to grid computing," Journal of Parallel and Distributed Computing, vol. 65, pp. 1533–1541, Dec. 2005.

- [178] C. Zhou, L.-T. Chia, and B.-S. Lee, “Service discovery and measurement based on daml-QoS ontology,” in Special Interest Tracks and Posters of the 14th International Conference on World Wide Web, WWW ’05, (New York, NY, USA), pp. 1070–1071, ACM, 2005.
- [179] D. K. Bimson, D. R. Hull, and D. Nieten, The Lexical Bridge: A Methodology for Bridging the Semantic Gaps between a Natural Language and an Ontology, pp. 137–151. Springer International Publishing, 2016.
- [180] B. D. Martino, G. Cretella, A. Esposito, A. Willner, A. Alloush, D. Bernstein, D. Vij, and J. Weinman, “Towards an ontology-based intercloud resource catalogue – the IEEE p2302 intercloud approach for a semantic resource exchange,” in Proceedings of the 2015 IEEE International Conference on Cloud Engineering, IC2E ’15, (Washington, DC, USA), pp. 458–464, IEEE Computer Society, 2015.
- [181] H. Ballani and P. Francis, “Towards a global ip anycast service,” SIGCOMM Comput. Commun. Rev., vol. 35, pp. 301–312, Aug. 2005.
- [182] T. Stevens, M. De Leenheer, C. Develder, F. De Turck, B. Dhoedt, and P. Demeester, “Astras: Architecture for scalable and transparent anycast services,” Communications and Networks, Journal of, vol. 9, no. 4, pp. 457–465, 2007.
- [183] G. P. Jesi, A. Montresor, and O. Babaoglu, “Proximity-aware superpeer overlay topologies,” in Proceedings of the Second IEEE International Conference on Self-Managed Networks, Systems, and Services, SelfMan’06, (Berlin, Heidelberg), pp. 43–57, Springer-Verlag, 2006.
- [184] M. Raack, D. Battré, A. Höing, and O. Kao, “Papnet: A proximity-aware alphanumeric overlay supporting ganesan on-line load balancing,” in Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems, ICPADS ’09, (Washington, DC, USA), pp. 440–447, IEEE Computer Society, 2009.
- [185] Y.-J. Joung, “Approaching neighbor proximity and load balance for range query in P2P networks,” Comput. Netw., vol. 52, pp. 1451–1472, May 2008.
- [186] Y.-J. Joung, W.-T. Wong, H.-M. Huang, and Y.-F. Chou, “Building a network-aware and load-balanced peer-to-peer system for range queries,” Comput. Netw., vol. 56, pp. 2148–2167, May 2012.
- [187] H. Jin, F. Luo, Q. Zhang, X. Liao, and H. Zhang, “Gtapestry: A locality-aware overlay network for high performance computing,” in Proceedings of the 11th IEEE Symposium on Computers and Communications, ISCC ’06, (Washington, DC, USA), pp. 76–81, IEEE Computer Society, 2006.
- [188] Y. Sun, L. Sun, X. Huang, and Y. Lin, “Resource discovery in locality-aware group-based semantic overlay of peer-to-peer networks,” in Proceedings of the 1st International Conference on Scalable Information Systems, InfoScale ’06, (New York, NY, USA), ACM, 2006.
- [189] J. Bo, “Heterogeneity-aware group-based semantic overlay network for P2P systems,” in Proceedings of the 2009 International Conference on Web Information Systems and

- Mining, WISM '09, (Washington, DC, USA), pp. 701–704, IEEE Computer Society, 2009.
- [190] X. Sun, K. Li, Y. Liu, and Y. Tian, “Slup: A semantic-based and location-aware unstructured P2P network,” in Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications, HPCC '08, (Washington, DC, USA), pp. 288–295, IEEE Computer Society, 2008.
 - [191] H. Ying, “Clsp2p: A P2P overlay combination location and semantic clustering,” in Proceedings of the 2010 Third International Symposium on Information Processing, ISIP '10, (Washington, DC, USA), pp. 143–147, IEEE Computer Society, 2010.
 - [192] J. Li, “Grid resource discovery based on semantically linked virtual organizations,” Future Gener. Comput. Syst., vol. 26, pp. 361–373, Mar. 2010.
 - [193] C. Xiao and L. Nianzu, “Overlay construction within diffserv domains for QoS-aware multicasting,” in Proceedings of the 2010 Third International Symposium on Information Science and Engineering, ISISE '10, (Washington, DC, USA), pp. 271–274, IEEE Computer Society, 2010.
 - [194] W. Li, Y. Wang, C. Li, S. Lu, and D. Chen, “A QoS-aware service selection algorithm for multimedia service overlay networks,” in Proceedings of the 13th International Conference on Parallel and Distributed Systems - Volume 01, ICPADS '07, (Washington, DC, USA), pp. 1–8, IEEE Computer Society, 2007.
 - [195] E. P. Duarte, L. Z. Granville, L. Pirmez, J. N. Souza, R. C. Andrade, L. Tarouco, R. B. Correia, and A. Lages, “Gigamanp2p: An overlay network for distributed QoS management and resilient routing,” Int. J. Netw. Manag., vol. 22, pp. 50–64, May 2011.
 - [196] M. Li and D. B. Hoang, “Fiac: A resource discovery-based two-level admission control for differentiated service networks,” Comput. Commun., vol. 28, pp. 2094–2104, Nov. 2005.
 - [197] W.-C. Chung, C.-J. Hsu, K.-C. Lai, K.-C. Li, and Y.-C. Chung, “Direction-aware resource discovery in large-scale distributed computing environments,” J. Supercomput., vol. 66, pp. 229–248, Oct. 2013.
 - [198] D. I. Wolinsky, Y. Liu, P. S. Juste, G. Venkatasubramanian, and R. Figueiredo, “On the design of scalable, self-configuring virtual networks,” in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, (New York, NY, USA), pp. 13:1–13:12, ACM, 2009.
 - [199] R. Ahmed, N. Limam, J. Xiao, Y. Iraqi, and R. Boutaba, “Resource and service discovery in large-scale multi-domain networks,” Communications Surveys Tutorials, IEEE, vol. 9, pp. 2–30, April 2007.
 - [200] J. Kempf and J. Goldschmidt, “Notification and subscription for slp,” March 2001. RFC 3082.
 - [201] Z. Xu, L. Chen, G. Gu, and C. Kruegel, “Peerpress: Utilizing enemies’ P2P strength against them,” in Proceedings of the 2012 ACM Conference on Computer

- and Communications Security, CCS '12, (New York, NY, USA), pp. 581–592, ACM, 2012.
- [202] A. Brocco, A. Malatras, and B. Hirsbrunner, “Proactive information caching for efficient resource discovery in a self-structured grid,” in Proceedings of the 2009 Workshop on Bio-inspired Algorithms for Distributed Systems, BADS '09, (New York, NY, USA), pp. 11–18, ACM, 2009.
 - [203] V. Iyengar, S. Tilak, M. J. Lewis, and N. B. Abu-Ghazaleh, “Non-uniform information dissemination for dynamic grid resource discovery,” in Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on, pp. 97–106, Aug 2004.
 - [204] S. Di and C.-L. Wang, “Decentralized proactive resource allocation for maximizing throughput of p2p grid,” J. Parallel Distrib. Comput., vol. 72, pp. 308–321, Feb. 2012.
 - [205] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, “A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks,” Computer Networks, vol. 52, no. 11, pp. 2097 – 2128, 2008.
 - [206] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, Artificial intelligence: a modern approach, vol. 74. Prentice hall Englewood Cliffs, 1995.
 - [207] H. Barjini, M. Othman, H. Ibrahim, and N. I. Udzir, “Shortcoming, problems and analytical comparison for flooding-based search techniques in unstructured p2p networks,” Peer-to-Peer Networking and Applications, vol. 5, no. 1, pp. 1–13, 2012.
 - [208] V. V. Dimakopoulos and E. Pitoura, “On the performance of flooding-based resource discovery,” Parallel and Distributed Systems, IEEE Transactions on, vol. 17, no. 11, pp. 1242–1252, 2006.
 - [209] R. Gaeta and M. Sereno, “On the evaluation of flooding-based search strategies in peer-to-peer networks,” Concurrency and Computation: Practice and Experience, vol. 20, no. 6, pp. 713–734, 2008.
 - [210] B. Awerbuch and R. G. Gallager, “A new distributed algorithm to find breadth first search trees,” IEEE Trans. Inf. Theor., vol. 33, pp. 315–322, May 1987.
 - [211] R. Zhou and E. A. Hansen, “Breadth-first heuristic search,” Artif. Intell., vol. 170, pp. 385–408, Apr. 2006.
 - [212] H. Shang and M. Kitsuregawa, “Efficient breadth-first search on large graphs with skewed degree distributions,” in Proceedings of the 16th International Conference on Extending Database Technology, EDBT '13, (New York, NY, USA), pp. 311–322, ACM, 2013.
 - [213] S. A. M. Makki and G. Havas, “Distributed algorithms for depth-first search,” Inf. Process. Lett., vol. 60, pp. 7–12, Oct. 1996.
 - [214] S. Makki and G. Havas, “Distributed algorithms for constructing a depth-first-search tree,” in Parallel Processing, 1994. Vol. 1. ICPP 1994. International Conference on, vol. 3, pp. 270–273, 1994.

- [215] C. Rhee, Y. D. Liang, S. K. Dhall, and S. Lakshmivarahan, "Efficient algorithms for finding depth-first and breadth-first search trees in permutation graphs," Inf. Process. Lett., vol. 49, pp. 45–50, Jan. 1994.
- [216] R. E. Korf, "Depth-limited search for real-time problem solving," Real-Time Systems, vol. 2, no. 1-2, pp. 7–24, 1990.
- [217] R. E. Korf, "Iterative-deepening-a: an optimal admissible tree search," in Proceedings of the 9th international joint conference on Artificial intelligence-Volume 2, pp. 1034–1036, Morgan Kaufmann Publishers Inc., 1985.
- [218] A. Felner, "Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm," in Fourth Annual Symposium on Combinatorial Search, 2011.
- [219] N. Bisnik and A. Abouzeid, "Modeling and analysis of random walk search algorithms in p2p networks," in Hot topics in peer-to-peer systems, 2005. HOT-P2P 2005. Second International Workshop on, pp. 95–103, IEEE, 2005.
- [220] N. Shenvi, J. Kempe, and K. B. Whaley, "Quantum random-walk search algorithm," Physical Review A, vol. 67, no. 5, p. 052307, 2003.
- [221] N. Bisnik and A. A. Abouzeid, "Optimizing random walk search algorithms in p2p networks," Computer Networks, vol. 51, no. 6, pp. 1499–1514, 2007.
- [222] M. Jelasity and M. van Steen, "Large-scale newscast computing on the Internet," Oct. 2002.
- [223] M. Zaharia and S. Keshav, "Gossip-based search selection in hybrid peer-to-peer networks," Concurrency and Computation: Practice and Experience, vol. 20, no. 2, pp. 139–153, 2008.
- [224] H. Chen, H. Jin, Y. Liu, and L. M. Ni, "Difficulty-aware hybrid search in peer-to-peer networks," Parallel and Distributed Systems, IEEE Transactions on, vol. 20, no. 1, pp. 71–82, 2009.
- [225] K. Oikonomou, D. Kogias, and I. Stavrakakis, "Probabilistic flooding for efficient information dissemination in random graph topologies," Comput. Netw., vol. 54, pp. 1615–1629, July 2010.
- [226] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," ACM Trans. Comput. Syst., vol. 17, pp. 41–88, May 1999.
- [227] U. Datta, A. Mukherjee, P. Sahu, and S. Kundu, "Resource utilization of multi-hop CDMA wireless sensor networks with efficient forwarding protocols," Procedia Engineering, vol. 64, pp. 46–55, 2013.
- [228] S. Ferretti, "Gossiping for resource discovering: An analysis based on complex network theory," Future Gener. Comput. Syst., vol. 29, pp. 1631–1644, Aug. 2013.
- [229] A. Ganesh, A.-M. Kermarrec, and L. Massoulie, "Peer-to-peer membership management for gossip-based protocols," Computers, IEEE Transactions on, vol. 52, no. 2, pp. 139–149, 2003.

- [230] S. Tang, E. Jaho, I. Stavrakakis, I. Koukoutsidis, and P. V. Mieghem, "Modeling gossip-based content dissemination and search in distributed networking," Comput. Commun., vol. 34, pp. 765–779, May 2011.
- [231] M. Jelasity, A. Montresor, and O. Babaoglu, "T-man: Gossip-based fast overlay topology construction," Comput. Netw., vol. 53, pp. 2321–2339, Aug. 2009.
- [232] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in Proceedings of the Eleventh International Conference on Information and Knowledge Management, CIKM '02, (New York, NY, USA), pp. 300–307, ACM, 2002.
- [233] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in Proceedings of the 16th International Conference on Supercomputing, ICS '02, (New York, NY, USA), pp. 84–95, ACM, 2002.
- [234] Y. Zhang and L. Liu, "Distance-aware bloom filters: Enabling collaborative search for efficient resource discovery," Future Gener. Comput. Syst., vol. 29, pp. 1621–1630, Aug. 2013.
- [235] S. Rhea and J. Kubiawicz, "Probabilistic location and routing," in INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 3, pp. 1248–1257 vol.3, 2002.
- [236] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on, pp. 23–32, 2002.
- [237] C. Yang and J. Wu, "Dominating-set-based searching in peer-to-peer networks," in Grid and Cooperative Computing (M. Li, X.-H. Sun, Q. ni Deng, and J. Ni, eds.), vol. 3032 of Lecture Notes in Computer Science, pp. 332–339, Springer Berlin Heidelberg, 2004.
- [238] S. Kutten, D. Peleg, and U. Vishkin, "Deterministic resource discovery in distributed networks," in Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '01, (New York, NY, USA), pp. 77–83, ACM, 2001.
- [239] B. Awerbuch and Y. Shiloach, "New connectivity and msf algorithms for shuffle-exchange network and pram," IEEE Trans. Comput., vol. 36, pp. 1258–1263, Oct. 1987.
- [240] I. Cidon, I. Gopal, and S. Kutten, "New models and algorithms for future networks," in Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC '88, (New York, NY, USA), pp. 79–89, ACM, 1988.
- [241] P. Chaudhuri, "An $O(\log n)$ parallel algorithm for strong connectivity augmentation problem," International Journal of Computer Mathematics, vol. 22, no. 3-4, pp. 187–197, 1987.
- [242] I. Abraham and D. Dolev, "Asynchronous resource discovery," in Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing, PODC '03, (New York, NY, USA), pp. 143–150, ACM, 2003.

- [243] S. Kutten and D. Peleg, “Asynchronous resource discovery in peer-to-peer networks,” Comput. Netw., vol. 51, pp. 190–206, Jan. 2007.
- [244] M. Pathan and R. Buyya, “Resource discovery and request-redirection for dynamic load sharing in multi-provider peering content delivery networks,” J. Netw. Comput. Appl., vol. 32, pp. 976–990, Sept. 2009.
- [245] T. Alam and Z. Raza, “An adaptive threshold based hybrid load balancing scheme with sender and receiver initiated approach using random information exchange,” Concurrency and Computation: Practice and Experience, 2016.
- [246] L. Watkins, R. Beyah, and C. Corbett, “Using network traffic to passively detect under utilized resources in high performance cluster grid computing environments,” in Proceedings of the First International Conference on Networks for Grid Applications, GridNets ’07, (ICST, Brussels, Belgium, Belgium), pp. 16:1–16:8, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.
- [247] C. Law, “An $o(\log n)$ randomized resource discovery algorithm (extended abstract),” 2000.
- [248] S. Kutten and D. Peleg, “Deterministic distributed resource discovery,” in Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, p. 336, ACM, 2000.
- [249] V. K. Garg and A. Aziz, “An efficient deterministic algorithm for the resource discovery problem,” 2000.
- [250] D. Tate, G. Steven, and F. Steven, “Static scheduling for out-of-order instruction issue processors,” in Computer Architecture Conference, 2000. ACAC 2000. 5th Australasian, pp. 90–96, IEEE, 2000.
- [251] A.-M. Kermarrec and M. van Steen, “Gossiping in distributed systems,” SIGOPS Oper. Syst. Rev., vol. 41, pp. 2–7, Oct. 2007.
- [252] G. D’Angelo, S. Ferretti, and M. Marzolla, “Adaptive event dissemination for peer-to-peer multiplayer online games,” in Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, SIMUTools ’11, (ICST, Brussels, Belgium, Belgium), pp. 312–319, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.
- [253] C. Georgiou, S. Gilbert, and D. R. Kowalski, “Meeting the deadline: On the complexity of fault-tolerant continuous gossip,” in Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC ’10, (New York, NY, USA), pp. 247–256, ACM, 2010.
- [254] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola, “Introducing reliability in content-based publish-subscribe through epidemic algorithms,” in Proceedings of the 2Nd International Workshop on Distributed Event-based Systems, DEBS ’03, (New York, NY, USA), pp. 1–8, ACM, 2003.

- [255] P. M. Melliar-Smith, L. E. Moser, I. Michel Lombera, and Y.-T. Chuang, “itrust: Trustworthy information publication, search and retrieval,” in Proceedings of the 13th International Conference on Distributed Computing and Networking, ICDCN’12, (Berlin, Heidelberg), pp. 351–366, Springer-Verlag, 2012.
- [256] E. Simonton, B. Choi, and S. Seidel, “Using gossip for dynamic resource discovery,” in Parallel Processing, 2006. ICPP 2006. International Conference on, pp. 319–328, 2006.
- [257] A. Brocco, A. Malatras, and B. Hirsbrunner, “Enabling efficient information discovery in a self-structured grid,” Future Gener. Comput. Syst., vol. 26, pp. 838–846, June 2010.
- [258] B. Akay and D. Karaboga, “A modified artificial bee colony algorithm for real-parameter optimization,” Inf. Sci., vol. 192, pp. 120–142, June 2012.
- [259] S. Ghosh and I. W. Marshall, “Simple model of collective decision making during nectar source selection by honey bees,” in CD Rom of Workshop on Memory and Learning Mechanisms in Autonomous Robots, 2005.
- [260] V. Tereshko, “Reaction-diffusion model of a honeybee colony’s foraging behaviour,” in Parallel Problem Solving from Nature PPSN VI (M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H.-P. Schwefel, eds.), vol. 1917 of Lecture Notes in Computer Science, pp. 807–816, Springer Berlin Heidelberg, 2000.
- [261] S. Y. Ko, I. Gupta, and Y. Jo, “A new class of nature-inspired algorithms for self-adaptive peer-to-peer computing,” ACM Trans. Auton. Adapt. Syst., vol. 3, pp. 11:1–11:34, Aug. 2008.
- [262] J. Taheri, Y. Choon Lee, A. Y. Zomaya, and H. J. Siegel, “A bee colony based optimization approach for simultaneous job scheduling and data replication in grid environments,” Comput. Oper. Res., vol. 40, pp. 1564–1578, June 2013.
- [263] M. Dorigo and G. Di Caro, “The ant colony optimization meta-heuristic,” in New Ideas in Optimization (D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, eds.), pp. 11–32, Maidenhead, UK, England: McGraw-Hill Ltd., UK, 1999.
- [264] M. Dorigo, V. Maniezzo, A. Colorni, M. Dorigo, M. Dorigo, V. Maniezzo, V. Maniezzo, A. Colorni, and A. Colorni, “Positive feedback as a search strategy,” tech. rep., No. 91-016, Politecnico di Milano, Italy, 1991.
- [265] K. Krynicki, J. Jaen, and J. A. Mocholi, “On the performance of ACo-based methods in P2P resource discovery,” Appl. Soft Comput., vol. 13, pp. 4813–4831, Dec. 2013.
- [266] T. Abdullah, A. Anjum, N. Bessis, S. Sotiriadis, and K. Bertels, “Nature inspired self organization for adhoc grids,” in Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, AINA ’13, (Washington, DC, USA), pp. 682–689, IEEE Computer Society, 2013.
- [267] Y. Deng, F. Wang, and A. Ciura, “Ant colony optimization inspired resource discovery in P2P grid systems,” J. Supercomput., vol. 49, pp. 4–21, July 2009.

- [268] E. Michlmayr, “Ant algorithms for search in unstructured peer-to-peer networks,” in Proceedings of the 22Nd International Conference on Data Engineering Workshops, ICDEW ’06, (Washington, DC, USA), pp. 142–, IEEE Computer Society, 2006.
- [269] G. C. Valdez, “A self-adaptive ant colony system for semantic query routing problem in p2p networks,” Computación y Sistemas, vol. 13, no. 4, pp. 433–448, 2010.
- [270] M. Dorigo and L. Gambardella, “Ant colony system: a cooperative learning approach to the traveling salesman problem,” Evolutionary Computation, IEEE Transactions on, vol. 1, no. 1, pp. 53–66, 1997.
- [271] T. Stützle and H. Hoos, “Improvements on the ant-system: Introducing the max-min ant system,” in Artificial Neural Nets and Genetic Algorithms, pp. 245–249, Springer Vienna, 1998.
- [272] H. Yousefipour and Z. Jafari, “Using neural search approach for resource discovery in P2P networks,” Procedia Computer Science, vol. 3, pp. 1512–1516, 2011.
- [273] S. Haykin, Neural Networks: A Comprehensive Foundation. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 1998.
- [274] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, and J. Vuori, “Resource discovery in P2P networks using evolutionary neural networks,” in International Conference on Advances in Intelligent Systems-Theory and Applications (AISTA 2004), 2004.
- [275] N. Ganguly, G. Canright, and A. Deutsch, “Design of an efficient search algorithm for P2P networks using concepts from natural immune systems,” in Parallel Problem Solving from Nature - PPSN VIII (X. Yao, E. Burke, J. Lozano, J. Smith, J. Merelo-Guervós, J. Bullinaria, J. Rowe, P. Tiño, A. Kabán, and H.-P. Schwefel, eds.), vol. 3242 of Lecture Notes in Computer Science, pp. 491–500, Springer Berlin Heidelberg, 2004.
- [276] S. Brown, “Microsoft’s viral search displays your content in real time as it spreads across twitter,” Science, 2014.
- [277] S. Goel, A. Anderson, J. Hofman, and D. J. Watts, “The structural virality of online diffusion,” Management Science, vol. 62, no. 1, pp. 180–196, 2016.
- [278] V. V. Dimakopoulos and E. Pitoura, “A peer-to-peer approach to resource discovery in multi-agent systems,” in Cooperative Information Agents VII, pp. 62–77, Springer, 2003.
- [279] F. Glover and M. Laguna, Tabu Search. Norwell, MA, USA: Kluwer Academic Publishers, 1997.
- [280] S. C. Yusta, “Different metaheuristic strategies to solve the feature selection problem,” Pattern Recogn. Lett., vol. 30, pp. 525–534, Apr. 2009.
- [281] S. Xu, Z. Ji, D. T. Pham, and F. Yu, “Bio-inspired binary bees algorithm for a two-level distribution optimisation problem,” Journal of Bionic Engineering, vol. 7, no. 2, pp. 161–167, 2010.

- [282] S. K. Dhurandher, S. Misra, P. Pruthi, S. Singhal, S. Aggarwal, and I. Woungang, “Using bee algorithm for peer-to-peer file searching in mobile ad hoc networks,” Journal of Network and Computer Applications, vol. 34, no. 5, pp. 1498–1508, 2011.
- [283] B. Segall and D. Arnold, “Elvin has left the building: A publish/subscribe notification service with quenching,” in Proceedings of AUUG97, pp. 3–5, Brisbane, Australia, 1997.
- [284] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, “Matching events in a content-based subscription system,” in Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC '99, (New York, NY, USA), pp. 53–61, ACM, 1999.
- [285] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, “Design and evaluation of a wide-area event notification service,” ACM Trans. Comput. Syst., vol. 19, pp. 332–383, Aug. 2001.
- [286] S. Sarat, V. Pappas, and A. Terzis, “On the use of anycast in dns,” in Computer Communications and Networks, 2006. ICCCN 2006. Proceedings.15th International Conference on, pp. 71–78, 2006.
- [287] T. Stevens, T. Wauters, C. Develder, F. De Turck, B. Dhoedt, and P. Demeester, “Analysis of an anycast based overlay system for scalable service discovery and execution,” Comput. Netw., vol. 54, pp. 97–111, Jan. 2010.
- [288] J. Seo, K. Cho, W. Cho, G. Park, and K. Han, “A discovery scheme based on carrier sensing in self-organizing bluetooth low energy networks,” Journal of Network and Computer Applications, vol. 65, pp. 72 – 83, 2016.
- [289] R. Dissanayaka, S. K. Prasad, S. B. Navathe, and V. Goyal, “Bsi: Bloom filter-based semantic indexing for unstructured p2p networks,” International Journal of Peer to Peer Networks, vol. 6, no. 1, p. 11, 2015.
- [290] R. Lima, C. Baquero, and H. Miranda, “Adaptive broadcast cancellation query mechanism for unstructured networks,” in Next Generation Mobile Applications, Services and Technologies, 2015 9th International Conference on, pp. 176–181, IEEE, 2015.
- [291] A. G. Medrano-Chávez, E. Pérez-Cortés, and M. Lopez-Guerrero, “A performance comparison of chord and kademia dhds in high churn scenarios,” Peer-to-Peer Networking and Applications, vol. 8, no. 5, pp. 807–821, 2015.
- [292] M. Atif and M. Mousavi, “Formal specification and analysis of accelerated heartbeat protocols,” in Proceedings of the 2010 Summer Computer Simulation Conference, SCSC '10, (San Diego, CA, USA), pp. 403–412, Society for Computer Simulation International, 2010.
- [293] E. Meshkova, J. Riihijärvi, and P. Mähönen, “Evaluation of dynamic query abolishment methods in heterogeneous networks,” in GLOBECOM, IEEE, 2006.
- [294] J. Zarrin, R. L. Aguiar, and J. P. Barraca, “Dynamic, scalable and flexible resource discovery for large-dimension many-core systems,” Future Generation Computer Systems, vol. 53, pp. 119 – 129, 2015.

- [295] J. Zarrin, R. L. Aguiar, and J. P. Barraca, “Elcore: Dynamic elastic resource management and discovery for future large-scale manycore enabled distributed systems,” Microprocessors and Microsystems, vol. 46, Part B, pp. 221 – 239, 2016.
- [296] V. Nagarajan and M. Mohamed, “A decentralized two phase resource discovery model for peer-to-peer grid environments,” Int J Adv Engg Tech/Vol. VII/Issue II/April-June, vol. 1092, p. 1095, 2016.
- [297] C. Pittaras, C. Papagianni, A. Leivadeas, P. Grosso, J. van der Ham, and S. Papavassiliou, “Resource discovery and allocation for federated virtualized infrastructures,” Future Generation Computer Systems, vol. 42, pp. 55 – 63, 2015.
- [298] Y. Zhang, Y. Jia, X. Huang, B. Zhou, and J. Gu, “A scalable method for efficient grid resource discovery,” in Cooperative Design, Visualization, and Engineering (Y. Luo, ed.), vol. 4674 of Lecture Notes in Computer Science, pp. 97–103, Springer Berlin Heidelberg, 2007.
- [299] A. Iamnitchi and I. T. Foster, “On fully decentralized resource discovery in grid environments,” in Proceedings of the Second International Workshop on Grid Computing, GRID ’01, (London, UK, UK), pp. 51–62, Springer-Verlag, 2001.
- [300] C. N. Ververidis and G. C. Polyzos, “Service discovery for mobile ad hoc networks: A survey of issues and techniques,” Commun. Surveys Tuts., vol. 10, pp. 30–45, July 2008.
- [301] M. H. Khoobkar and M. Mahdavi, “Enabling efficient peer to peer resource discovery in dynamic grids using variable size routing indexes,” in Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks, ISPAN ’09, (Washington, DC, USA), pp. 691–695, IEEE Computer Society, 2009.
- [302] F. Butt, S. S. Bokhari, A. Abhari, and A. Ferworn, “Scalable grid resource discovery through distributed search,” arXiv preprint arXiv:1110.1685, 2011.
- [303] A. Passarella, “Review: A survey on content-centric technologies for the current internet: CDn and P2P solutions,” Comput. Commun., vol. 35, pp. 1–32, Jan. 2012.
- [304] M. Klusch, “Information agent technology for the internet: A survey,” Data Knowl. Eng., vol. 36, pp. 337–372, Mar. 2001.
- [305] M. Hussin, N. A. W. A. Hamid, and K. A. Kasmiran, “Improving reliability in resource management through adaptive reinforcement learning for distributed systems,” Journal of Parallel and Distributed Computing, vol. 75, pp. 93 – 100, 2015.
- [306] A. Hameurlain and F. Morvan, Transactions on Large-Scale Data- and Knowledge-Centered Systems I, ch. Evolution of Query Optimization Methods, pp. 211–242. Berlin, Heidelberg: Springer-Verlag, 2009.
- [307] S. Ghamri-Doudane and N. Agoulmine, “Enhanced DHT-based P2P architecture for effective resource discovery and management,” J. Netw. Syst. Manage., vol. 15, pp. 335–354, Sept. 2007.

- [308] D. Lazaro, J. Marques, and X. Vilajosana, “Flexible resource discovery for decentralized P2P and volunteer computing systems,” in Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on, pp. 235–240, June 2010.
- [309] R. Baldoni, S. Bonomi, A. Cerocchi, and L. Querzoni, “Virtual tree: A robust architecture for interval valid queries in dynamic distributed systems,” J. Parallel Distrib. Comput., vol. 73, pp. 1135–1145, Aug. 2013.
- [310] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: A resilient global-scale overlay for service deployment,” Selected Areas in Communications, IEEE Journal on, vol. 22, no. 1, pp. 41–53, 2004.
- [311] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil, “A performance vs. cost framework for evaluating DHT design tradeoffs under churn,” in INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, vol. 1, pp. 225–236, IEEE, 2005.
- [312] J. Risson and T. Moors, “Survey of research towards robust peer-to-peer networks: Search methods,” Comput. Netw., vol. 50, pp. 3485–3521, Dec. 2006.
- [313] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, “Handling churn in a DHT,” in Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC ’04, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2004.
- [314] M. Castro, M. Costa, and A. Rowstron, “Performance and dependability of structured peer-to-peer overlays,” in Proceedings of the 2004 International Conference on Dependable Systems and Networks, DSN ’04, (Washington, DC, USA), pp. 9–, IEEE Computer Society, 2004.
- [315] D. Liben-Nowell, H. Balakrishnan, and D. Karger, “Analysis of the evolution of peer-to-peer systems,” in Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing, PODC ’02, (New York, NY, USA), pp. 233–242, ACM, 2002.
- [316] S. Di, C. L. Wang, W. Zhang, and L. Cheng, “Probabilistic best-fit multi-dimensional range query in self-organizing cloud,” in 2011 International Conference on Parallel Processing, pp. 763–772, Sept 2011.
- [317] J.-S. Kim, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman, “Using content-addressable networks for load balancing in desktop grids,” in Proceedings of the 16th International Symposium on High Performance Distributed Computing, HPDC ’07, (New York, NY, USA), pp. 189–198, ACM, 2007.
- [318] F. Falchi, C. Gennaro, and P. Zezula, “Nearest neighbor search in metric spaces through content-addressable networks,” Inf. Process. Manage., vol. 44, pp. 411–429, Jan. 2008.
- [319] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer, “Range queries in trie-structured overlays,” in Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing, P2P ’05, (Washington, DC, USA), pp. 57–66, IEEE Computer Society, 2005.

- [320] S. Di, C.-L. Wang, W. Zhang, and L. Cheng, “Probabilistic best-fit multi-dimensional range query in self-organizing cloud,” in Proceedings of the 2011 International Conference on Parallel Processing, ICPP ’11, (Washington, DC, USA), pp. 763–772, IEEE Computer Society, 2011.
- [321] M. Cai, M. Frank, J. Chen, and P. Szekely, “Maan: A multi-attribute addressable network for grid information services,” in Proceedings of the 4th International Workshop on Grid Computing, GRID ’03, (Washington, DC, USA), pp. 184–, IEEE Computer Society, 2003.
- [322] A. R. Bharambe, M. Agrawal, and S. Seshan, “Mercury: Supporting scalable multi-attribute range queries,” SIGCOMM Comput. Commun. Rev., vol. 34, pp. 353–366, Aug. 2004.
- [323] D. Li, J. Cao, X. Lu, and K. C. C. Chen, “Efficient range query processing in peer-to-peer systems,” IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 1, pp. 78–91, 2009.
- [324] P. Ganesan, B. Yang, and H. Garcia-Molina, “One torus to rule them all: Multi-dimensional queries in P2P systems,” in Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004, WebDB ’04, (New York, NY, USA), pp. 19–24, ACM, 2004.
- [325] A. González-Beltrán, P. Milligan, and P. Sage, “Range queries over skip tree graphs,” Comput. Commun., vol. 31, pp. 358–374, Feb. 2008.
- [326] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson, “Design and implementation trade-offs for wide-area resource discovery,” ACM Trans. Internet Technol., vol. 8, pp. 18:1–18:44, Oct. 2008.
- [327] S. Basu, S. Banerjee, P. Sharma, and S.-J. Lee, “Nodewiz: peer-to-peer resource discovery for grids,” in Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on, vol. 1, pp. 213–220, May 2005.
- [328] C. Zhang, W. Xiao, D. Tang, and J. Tang, “P2P-based multidimensional indexing methods: A survey,” J. Syst. Softw., vol. 84, pp. 2348–2362, Dec. 2011.
- [329] F. Buccafurri, F. Furfaro, G. M. Mazzeo, and D. Saccí, “A quad-tree based multiresolution approach for two-dimensional summary data,” Inf. Syst., vol. 36, pp. 1082–1103, Nov. 2011.
- [330] D. Chatziantoniou and A. Anagnostopoulos, “Hierarchical stream aggregates: Querying nested stream sessions,” in Proceedings of the 16th International Conference on Scientific and Statistical Database Management, SSDBM ’04, (Washington, DC, USA), pp. 439–, IEEE Computer Society, 2004.
- [331] D. Kostoulas, D. Psaltoulis, I. Gupta, K. P. Birman, and A. J. Demers, “Active and passive techniques for group size estimation in large-scale and dynamic distributed systems,” J. Syst. Softw., vol. 80, pp. 1639–1658, Oct. 2007.

- [332] R. Van Renesse, K. P. Birman, and W. Vogels, “Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining,” ACM Trans. Comput. Syst., vol. 21, pp. 164–206, May 2003.
- [333] P. Cao and Z. Wang, “Efficient top-k query calculation in distributed networks,” in Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing, PODC ’04, (New York, NY, USA), pp. 206–215, ACM, 2004.
- [334] P. Reynolds and A. Vahdat, “Efficient peer-to-peer keyword searching,” in Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, Middleware ’03, (New York, NY, USA), pp. 21–40, Springer-Verlag New York, Inc., 2003.
- [335] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making gnutella-like P2P systems scalable,” in Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM ’03, (New York, NY, USA), pp. 407–418, ACM, 2003.
- [336] A. S. Tigelaar, D. Hiemstra, and D. Trieschnigg, “Peer-to-peer information retrieval: An overview,” ACM Trans. Inf. Syst., vol. 30, pp. 9:1–9:34, May 2012.
- [337] S. Ghamri-Doudane and N. Agoulmine, “Enhancing the P2P protocols to support advanced multi-keyword queries,” in Proceedings of the 5th International IFIP-TC6 Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems, NETWORKING’06, (Berlin, Heidelberg), pp. 630–641, Springer-Verlag, 2006.
- [338] M. Steiner, E. W. Biersack, and T. En Najjary, “Actively monitoring peers in KAD,” in IPTPS 2007, 6th International Workshop on Peer-to-Peer Systems, February 26-27, 2007, Bellevue, USA, (Bellevue, UNITED STATES), Feb. 2007.
- [339] D. Carra and E. W. Biersack, “Building a reliable P2P system out of unreliable P2P clients: The case of kad,” in Proceedings of the 2007 ACM CoNEXT Conference, CoNEXT ’07, (New York, NY, USA), pp. 28:1–28:12, ACM, 2007.
- [340] M. Steiner, T. En-Najjary, and E. W. Biersack, “Long term study of peer behavior in the kad DHT,” IEEE/ACM Trans. Netw., vol. 17, pp. 1371–1384, Oct. 2009.
- [341] J. Li, B. Loo, J. Hellerstein, M. Kaashoek, D. Karger, and R. Morris, On the Feasibility of Peer-to-Peer Web Indexing and Search, vol. 2735 of Lecture Notes in Computer Science, pp. 207–215. Springer Berlin Heidelberg, 2003.
- [342] Z. Gao, X. Li, L. Wang, J. Zhao, Y. Zhao, and H. Shi, “Bfgsdp: Bloom filter guided service discovery protocol for MANETs,” in Proceedings of the 20th International Teletraffic Conference on Managing Traffic Performance in Converged Networks, ITC20’07, (Berlin, Heidelberg), pp. 446–457, Springer-Verlag, 2007.
- [343] S. Cheng, C. K. Chang, and L.-J. Zhang, “An efficient service discovery algorithm for counting bloom filter-based service registry,” in Proceedings of the 2009 IEEE International Conference on Web Services, ICWS ’09, (Washington, DC, USA), pp. 157–164, IEEE Computer Society, 2009.

- [344] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in 2008 Grid Computing Environments Workshop, pp. 1–10, Nov 2008.
- [345] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Distributed resource discovery on planetlab with SWord," in Proceedings of the ACM/USENIX Workshop on Real, Large Distributed Systems (WORLDS), 2004.
- [346] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and implementation tradeoffs for wide-area resource discovery," in High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on, pp. 113–124, July 2005.
- [347] H. M. N. D. Bandara and A. P. Jayasumana, "Characteristics of multi-attribute resources/queries and implications on P2P resource discovery," in Proceedings of the 2011 9th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA '11, (Washington, DC, USA), pp. 173–180, IEEE Computer Society, 2011.
- [348] S. Basu, L. Costa, F. Brasileiro, S. Banerjee, P. Sharma, and S.-J. Lee, "Nodewiz: Fault-tolerant grid information service," Peer-to-Peer Networking and Applications, vol. 2, no. 4, pp. 348–366, 2009.
- [349] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in Job Scheduling Strategies for Parallel Processing, pp. 44–60, Springer, 2003.
- [350] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in Job Scheduling Strategies for Parallel Processing, pp. 44–60, Springer, 2003.
- [351] S. Zhou, "LSf: Load sharing in large heterogeneous distributed systems," in I Workshop on Cluster Computing, 1992.
- [352] R. L. Henderson, "Job scheduling under the portable batch system," in Job scheduling strategies for parallel processing, pp. 279–294, Springer, 1995.
- [353] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. F. Skovira, "Workload management with loadleveler," IBM Redbooks, vol. 2, p. 2, 2001.
- [354] A. Keller and A. Reinefeld, "Ccs resource management in networked hpc systems," in Proceedings of the Seventh Heterogeneous Computing Workshop, HCW '98, (Washington, DC, USA), pp. 44–, IEEE Computer Society, 1998.
- [355] W. G. S. Microsystems), "Sun grid engine: Towards creating a compute power grid," in Proceedings of the 1st International Symposium on Cluster Computing and the Grid, CCGRID '01, (Washington, DC, USA), pp. 35–, IEEE Computer Society, 2001.
- [356] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," in Distributed Computing Systems, 1988., 8th International Conference on, pp. 104–111, IEEE, 1988.
- [357] J. Basney, M. Livny, and T. Tannenbaum, "Deploying a high throughput computing cluster," High performance cluster computing, vol. 1, no. 5, pp. 356–361, 1999.

- [358] B. N. Chun and D. E. Culler, “Market-based proportional resource sharing for clusters,” tech. rep., University of California at Berkeley, Berkeley, CA, USA, 2000.
- [359] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, “Tycoon: An implementation of a distributed, market-based resource allocation system,” Multiagent Grid Syst., vol. 1, pp. 169–182, Aug. 2005.
- [360] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, “Dynamic virtual clusters in a grid site manager,” in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, HPDC ’03, (Washington, DC, USA), pp. 90–, IEEE Computer Society, 2003.
- [361] R. Buyya, D. Abramson, and J. Giddy, “Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid,” in High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on, vol. 1, pp. 283–289, IEEE, 2000.
- [362] F. Kon, R. H. Campbell, M. D. Mickunas, and K. Nahrstedt, “2k: A distributed operating system for dynamic heterogeneous environments,” tech. rep., University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1999.
- [363] L. Bölöni, K. Jun, K. Palacz, R. Sion, and D. C. Marinescu, “The bond agent system and applications,” in Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000, (London, UK, UK), pp. 99–112, Springer-Verlag, 2000.
- [364] P. Chandra, Y. H. Chu, A. Fisher, J. Gao, C. Kosak, T. S. Ng, P. Steenkiste, E. Takahashi, and H. Zhang, “Darwin: Customizable resource management for value-added network services,” Netwrk. Mag. of Global Internetwkg., vol. 15, pp. 22–35, Jan. 2001.
- [365] M. Ripeanu, M. Bowman, J. S. Chase, I. Foster, and M. Milenkovic, “Globus and planetlab resource management solutions compared,” in Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, HPDC ’04, (Washington, DC, USA), pp. 246–255, IEEE Computer Society, 2004.
- [366] S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw, “The legion resource management system,” in Proceedings of the Job Scheduling Strategies for Parallel Processing, IPSP/SPDP ’99/JSSPP ’99, (London, UK, UK), pp. 162–178, Springer-Verlag, 1999.
- [367] H. Nakada, M. Sato, and S. Sekiguchi, “Design and implementations of ninf: Towards a global computing infrastructure,” Future Gener. Comput. Syst., vol. 15, pp. 649–658, Oct. 1999.
- [368] A. Bradley, K. Curran, and G. Parr, “Discovering resources in computational grid environments,” The Journal of Supercomputing, vol. 35, no. 1, pp. 27–49, 2006.
- [369] T. S. Somasundaram and K. Govindarajan, “Cloudrb: A framework for scheduling and managing high-performance computing (HPC) applications in science cloud,” Future Generation Computer Systems, vol. 34, no. 0, pp. 47–65, 2014. Special Section: Distributed Solutions for Ubiquitous Computing and Ambient Intelligence.

- [370] J. A. Wickboldt, R. P. Esteves, M. B. de Carvalho, and L. Z. Granville, “Resource management in iaas cloud platforms made flexible through programmability,” Computer Networks, vol. 68, no. 0, pp. 54–70, 2014. Communications and Networking in the Cloud.
- [371] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The eucalyptus open-source cloud-computing system,” in Cluster Computing and the Grid, 2009. CCGRID ’09. 9th IEEE/ACM International Symposium on, pp. 124–131, May 2009.
- [372] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, “From virtualized resources to virtual computing grids: the in-vigo system,” Future Generation Computer Systems, vol. 21, no. 6, pp. 896 – 909, 2005.
- [373] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle, “Dynamic virtual clusters in a grid site manager,” in High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on, pp. 90–100, June 2003.
- [374] Y. Jiang, X. Shen, J. Chen, and R. Tripathi, “Analysis and approximation of optimal co-scheduling on chip multiprocessors,” in Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT ’08, (New York, NY, USA), pp. 220–229, ACM, 2008.
- [375] D.-W. Kim, K.-H. Kim, W. Jang, and F. F. Chen, “Unrelated parallel machine scheduling with setup times using simulated annealing,” Robotics and Computer-Integrated Manufacturing, vol. 18, no. 3–4, pp. 223 – 231, 2002. 11th International Conference on Flexible Automation and Intelligent Manufacturing.
- [376] J. E. Boillat and P. G. Kropf, “A fast distributed mapping algorithm,” in Proceedings of the Joint International Conference on Vector and Parallel Processing, CONPAR 90/VAPP IV, (London, UK, UK), pp. 405–416, Springer-Verlag, 1990.
- [377] C. Marcon, E. Moreno, N. Calazans, and F. Moraes, “Evaluation of algorithms for low energy mapping onto nocs,” in Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on, pp. 389–392, May 2007.
- [378] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, “Seec: A framework for self-aware management of multicore resources,” tech. rep., Massachusetts Institute of Technology, 2011.
- [379] H. Shojaei, A.-H. Ghamarian, T. Basten, M. Geilen, S. Stuijk, and R. Hoes, “A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management,” in Design Automation Conference, 2009. DAC ’09. 46th ACM/IEEE, pp. 917–922, July 2009.
- [380] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele, “Scenario-based design flow for mapping streaming applications onto on-chip many-core systems,” in Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES ’12, (New York, NY, USA), pp. 71–80, ACM, 2012.

- [381] B. Yang, L. Guang, T. Säntti, and J. Plosila, “Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip,” Microprocessors and Microsystems, vol. 37, no. 4–5, pp. 460 – 471, 2013.
- [382] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, and D. Soudris, “Distributed run-time resource management for malleable applications on many-core platforms,” in Proceedings of the 50th Annual Design Automation Conference, DAC ’13, (New York, NY, USA), pp. 168:1–168:6, ACM, 2013.
- [383] F. Dong and S. G. Akl, “Scheduling algorithms for grid computing: State of the art and open problems,” tech. rep., Technical report, 2006.
- [384] G. Sabin, M. Lang, and P. Sadayappan, “Moldable parallel job scheduling using job efficiency: An iterative approach,” in Job Scheduling Strategies for Parallel Processing (E. Frachtenberg and U. Schwiegelshohn, eds.), vol. 4376 of Lecture Notes in Computer Science, pp. 94–114, Springer Berlin Heidelberg, 2007.
- [385] M. D. Ciletti, Advanced Digital Design with the Verilog HDL. Upper Saddle River, NJ, USA: Prentice Hall Press, 2nd ed., 2010.
- [386] E. D. Reilly, Milestones in Computer Science and Information Technology. Westport, CT, USA: Greenwood Publishing Group Inc., 2003.
- [387] D. J. Smith, HDL Chip Design: A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs Using VHDL or Verilog. Doone Publications, 1998.
- [388] X.-H. Sun and Y. Chen, “Reevaluating amdahl’s law in the multicore era,” Journal of Parallel and Distributed Computing, vol. 70, no. 2, pp. 183–188, 2010.
- [389] P. Jacob, A. Zia, O. Erdogan, P. M. Belemjian, J.-W. Kim, M. Chu, R. P. Kraft, J. F. McDonald, and K. Bernstein, “Mitigating memory wall effects in high-clock-rate and multicore cmos 3-d processor memory stacks,” Proceedings of the IEEE, vol. 97, no. 1, pp. 108–122, 2009.
- [390] X.-H. Sun, Y. Chen, and S. Byna, “Scalable computing in the multicore era,” in Proceedings of the International Symposium on Parallel Architectures, Algorithms and Programming, Citeseer, 2008.
- [391] W. W. Gropp and E. L. Lusk, “A taxonomy of programming models for symmetric multiprocessors and smp clusters,” in Proceedings of the Conference on Programming Models for Massively Parallel Computers, PMMP ’95, (Washington, DC, USA), pp. 2–, IEEE Computer Society, 1995.
- [392] F. P. Miller, A. F. Vandome, and J. McBrewster, Huffman Coding: Computer Science, Algorithm, Lossless Data Compression, Variable- Length Code, David A. Huffman, Doctor of Philosophy, Massachusetts Institute of Technology. Alpha Press, 2009.
- [393] F. P. Miller, A. F. Vandome, and J. McBrewster, Lossless Data Compression: Data Compression, Algorithm, Lossy Compression, Bit Rate, ZIP (File Format), Unix, Gzip, Portable Network Graphics, Graphics Interchange Format, Tagged Image File Format. Alpha Press, 2009.

- [394] J.-L. Zhou and Y. Fu, "Scientific data lossless compression using fast neural network," in Proceedings of the Third International Conference on Advances in Neural Networks Volume Part I, ISNN'06, (Berlin, Heidelberg), pp. 1293–1298, Springer-Verlag, 2006.
- [395] M.-B. Lin and Y.-Y. Chang, "A new architecture of a two-stage lossless data compression and decompression algorithm," IEEE Trans. Very Large Scale Integr. Syst., vol. 17, pp. 1297–1303, Sept. 2009.
- [396] B. Alik and N. Lukač, "Chain code lossless compression using move-to-front transform and adaptive run-length encoding," Image Commun., vol. 29, pp. 96–106, Jan. 2014.
- [397] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," ACM Trans. Inf. Syst., vol. 16, pp. 256–294, July 1998.
- [398] F. M. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: basic properties," Information Theory, IEEE Transactions on, vol. 41, no. 3, pp. 653–664, 1995.
- [399] P. Fenwick, "Burrows–wheeler compression: Principles and reflections," Theor. Comput. Sci., vol. 387, pp. 200–219, Nov. 2007.
- [400] L. L. Larmore and D. S. Hirschberg, "A fast algorithm for optimal length-limited huffman codes," Journal of the ACM (JACM), vol. 37, no. 3, pp. 464–473, 1990.
- [401] F. J. Seinstra, J. Maassen, R. V. van Nieuwpoort, N. Drost, T. van Kessel, B. van Werkhoven, J. Urbani, C. Jacobs, T. Kielmann, and H. E. Bal, Grids, Clouds and Virtualization, ch. Jungle Computing: Distributed Supercomputing Beyond Clusters, Grids, and Clouds, pp. 167–197. London: Springer London, 2011.
- [402] M. Hajibaba and S. Gorgin, "A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing," CIT. Journal of Computing and Information Technology, vol. 22, no. 2, pp. 69–84, 2014.
- [403] Y. Wang, T. Uehara, and R. Sasaki, "Fog computing: Issues and challenges in security and forensics," in Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual, vol. 3, pp. 53–59, IEEE, 2015.
- [404] R. S. Segall and N. Gupta, "Overview of global supercomputing," Research and Applications in Global Supercomputing, p. 1, 2015.
- [405] E. Jeannot and J. Zilinskas, High-performance Computing on Complex Environments, vol. 96. John Wiley & Sons, 2014.
- [406] Y.-C. Wu, C.-M. Liu, and J.-H. Wang, "Enhancing the performance of locating data in chord-based P2P systems," in Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on, pp. 841–846, Dec. 2008.
- [407] B. Wiley, "Distributed hash tables, part i," Linux J., vol. 2003, pp. 7–, Oct. 2003.
- [408] J. Kuntraruk, Application Resource Requirement Estimation in a Parallel-pipeline Model of Execution on a Computational Grid. PhD thesis, Lehigh University, Bethlehem, PA, USA, 2003. AAI3117162.

- [409] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauer, R. Subramonian, and T. von Eicken, "Logp: A practical model of parallel computation," Commun. ACM, vol. 39, pp. 78–85, Nov. 1996.
- [410] C. Partridge, T. Mendez, and W. Milliken, "Host anycasting service," 1993. RFC 1546.
- [411] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei, "Application-layer anycasting," in INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE, vol. 3, pp. 1388–1396vol.3, Apr. 1997.
- [412] F. Hao, E. W. Zegura, and M. H. Ammar, "QoS routing for anycast communications: motivation and an architecture for diffserv networks," Communications Magazine, IEEE, vol. 40, no. 6, pp. 48–56, 2002.
- [413] W. M. Arden, "The international technology roadmap for semiconductors—perspectives and challenges for the next 15 years," Current Opinion in Solid State and Materials Science, vol. 6, no. 5, pp. 371–377, 2002.
- [414] I. K. Kim, J. Steele, Y. Qi, and M. Humphrey, "Comprehensive elastic resource management to ensure predictable performance for scientific applications on public iaas clouds," in Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14, (Washington, DC, USA), pp. 355–362, IEEE Computer Society, 2014.
- [415] T. Desell, K. E. Maghraoui, and C. A. Varela, "Malleable applications for scalable high performance computing," Cluster Computing, vol. 10, no. 3, pp. 323–337, 2007.
- [416] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel, "Distrm: distributed resource management for on-chip many-core systems," in Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 119–128, ACM, 2011.
- [417] B. Saha, A.-R. Adl-Tabatabai, A. Ghuloum, M. Rajagopalan, R. L. Hudson, L. Petersen, V. Menon, B. Murphy, T. Shpeisman, E. Sprangle, A. Rohillah, D. Carmean, and J. Fang, "Enabling scalability and performance in a large scale cmp environment," SIGOPS Oper. Syst. Rev., vol. 41, pp. 73–86, Mar. 2007.
- [418] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, et al., "Invasive manycore architectures," in ASP-DAC, pp. 193–200, 2012.
- [419] A. Schüpbach, S. Peter, A. Baumann, T. Roscoe, P. Barham, T. Harris, and R. Isaacs, "Embracing diversity in the barrellish manycore operating system," in Proceedings of the Workshop on Managed Many-Core Systems, p. 27, Association for Computing Machinery, Inc., June 2008.
- [420] R. Marculescu, U. Y. Ogras, L. S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, pp. 3–21, Jan 2009.

- [421] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang, "Mapping of applications to mpsoes," in Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '11, (New York, NY, USA), pp. 109–118, ACM, 2011.
- [422] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in Proceedings of the 50th Annual Design Automation Conference, DAC '13, (New York, NY, USA), pp. 1:1–1:10, ACM, 2013.
- [423] H.-S. Wu, C.-J. Wang, and J.-Y. Xie, "Terascaler elb-an algorithm of prediction-based elastic load balancing resource management in cloud computing," in Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on, pp. 649–654, IEEE, 2013.
- [424] U. Sharma, Elastic resource management in cloud computing platforms. PhD thesis, University of Massachusetts, Amherst, Jan. 2013.
- [425] M. Kesavan, A. Gavrilovska, and K. Schwan, "Elastic resource allocation in datacenters: Gremlins in the management plane," ELASTIC, vol. 1, p. 8, 2012.
- [426] T. Erl, R. Puttini, and Z. Mahmood, Cloud computing: concepts, technology, & architecture. Pearson Education, 2013.
- [427] I. K. Kim, J. Steele, Y. Qi, and M. Humphrey, "Comprehensive elastic resource management to ensure predictable performance for scientific applications on public iaas clouds," in Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on, pp. 355–362, IEEE, 2014.
- [428] L. Schubert, A. Kipp, and S. Wesner, "Above the clouds: From grids to service-oriented operating systems.," in Future Internet Assembly, pp. 238–249, 2009.
- [429] J. Zarrin, R. L. Aguiar, and J. P. Barraca, "A self-organizing and self-configuration algorithm for resource management in service-oriented systems," in Computers and Communication (ISCC), 2014 IEEE Symposium on, pp. 1–7, June 2014.
- [430] J. Lelli, D. Faggioli, T. Cucinotta, and G. Lipari, "An experimental comparison of different real-time schedulers on multicore systems," J. Syst. Softw., vol. 85, pp. 2405–2416, Oct. 2012.
- [431] K. G. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer science and engineering," Proceedings of the IEEE, vol. 82, pp. 6–24, Jan. 1994.
- [432] E. G. C. Jr., M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling.," SIAM J. Comput., vol. 7, no. 1, pp. 1–17, 1978.
- [433] S. Castano, A. Ferrara, S. Montanelli, and D. Zucchelli, "Helios: a general framework for ontology-based knowledge sharing and evolution in P2P systems," in Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on, pp. 597–603, Sept. 2003.

- [434] K. Sripanidkulchai and H. Zhang, "Content location in peer-to-peer systems: Exploiting locality," in Web Content Delivery (X. Tang, J. Xu, and S. Chanson, eds.), vol. 2 of Web Information Systems Engineering and Internet Technologies Book Series, pp. 73–97, Springer US, 2005.
- [435] N. Bisnik and A. A. Abouzeid, "Optimizing random walk search algorithms in {P2P} networks," Computer Networks, vol. 51, no. 6, pp. 1499–1514, 2007.
- [436] J. Li and R. Yahyapour, "Learning-based negotiation strategies for grid scheduling," in Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on, vol. 1, pp. 8 pp.–583, May 2006.
- [437] A. Sharma and S. Bawa, "Comparative analysis of resource discovery approaches in grid computing," Journal of Computers, vol. 3, no. 5, pp. 60–64, 2008.
- [438] I. Filali, F. Huet, and C. Vergoni, "A simple cache based mechanism for peer to peer resource discovery in grid environments," in Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on, pp. 602–608, May 2008.
- [439] A. Iamnitchi, I. Foster, and D. Nurmi, "A peer-to-peer approach to resource location in grid environments," in High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on, pp. 419–, 2002.
- [440] A. Iamnitchi and I. Foster, "A peer-to-peer approach to resource location in grid environments," in Grid Resource Management (J. Nabrzyski and J. Schopf, Jennifer M. Weglarz, eds.), vol. 64 of International Series on Operations Research and Management Science, pp. 413–429, Springer US, 2004.
- [441] C. Papadakis, P. Fragopoulou, E. Markatos, E. Athanasopoulos, M. Dikaiakos, and A. Labrinidis, "A feedback-based approach to reduce duplicate messages in unstructured peer-to-peer networks," in Integrated Research in GRID Computing (S. Gorlatch and M. Danelutto, eds.), pp. 103–118, Springer US, 2007.
- [442] E. Pournaras, G. Exarchakos, and N. Antonopoulos, "Load-driven neighbourhood reconfiguration of gnutella overlay," Computer Communications, vol. 31, no. 13, pp. 3030–3039, 2008. Special Issue: Self-organization and self-management in communications as applied to autonomic networks.
- [443] A. Furno and E. Zimeo, "Self-scaling cooperative discovery of service compositions in unstructured {P2P} networks," Journal of Parallel and Distributed Computing, vol. 74, no. 10, pp. 2994–3025, 2014.
- [444] E. Jeanvoine and C. Morin, "Rw-ogs: An optimized randomwalk protocol for resource discovery in large scale dynamic grids," in Grid Computing, 2008 9th IEEE/ACM International Conference on, pp. 168–175, Sept. 2008.
- [445] R. Robinson and J. Indulska, "The emergence of order in random walk resource discovery protocols," in Knowledge-Based Intelligent Information and Engineering Systems (R. Khosla, R. Howlett, and L. Jain, eds.), vol. 3683 of Lecture Notes in Computer Science, pp. 827–833, Springer Berlin Heidelberg, 2005.

- [446] V. Bioglio, R. Gaeta, M. Grangetto, and M. Sereno, “Rateless codes and random walks for P2P resource discovery in grids,” Parallel and Distributed Systems, IEEE Transactions on, vol. 25, pp. 1014–1023, Apr. 2014.
- [447] D. Zhou and V. Lo, “Cluster computing on the fly: resource discovery in a cycle sharing peer-to-peer system,” in Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on, pp. 66–73, Apr. 2004.
- [448] N. Bisnik and A. Abouzeid, “Modeling and analysis of random walk search algorithms in P2P networks,” in Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on, pp. 95–103, July 2005.
- [449] S. El-Ansary, L. Alima, P. Brand, and S. Haridi, “Efficient broadcast in structured P2P networks,” in Peer-to-Peer Systems II (M. Kaashoek and I. Stoica, eds.), vol. 2735 of Lecture Notes in Computer Science, pp. 304–314, Springer Berlin Heidelberg, 2003.
- [450] M. Sharmin, S. Ahmed, and S. Ahamed, “Safe-rd (secure, adaptive, fault tolerant, and efficient resource discovery) in pervasive computing environments,” in Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on, vol. 2, pp. 271–276 Vol. 2, Apr. 2005.
- [451] J. Zarrin, R. L. Aguiar, and J. P. Barraca, “A self-organizing and self-configuration algorithm for resource management in service-oriented systems,” in 19th IEEE Symposium on Computers and Communications (IEEE ISCC 2014), (Madeira, Portugal), June 2014.
- [452] J. F. Perez, G. Casale, and S. Pacheco-Sanchez, “Estimating computational requirements in multi-threaded applications,” Software Engineering, IEEE Transactions on, vol. 41, no. 3, pp. 264–278, 2015.
- [453] W. Wang and G. Casale, “Bayesian service demand estimation using gibbs sampling,” in Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on, pp. 567–576, IEEE, 2013.
- [454] A. Kalbasi, D. Krishnamurthy, J. Rolia, and S. Dawson, “Dec: Service demand estimation with confidence,” Software Engineering, IEEE Transactions on, vol. 38, no. 3, pp. 561–578, 2012.
- [455] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, “Quality-of-service in cloud computing: modeling techniques and their applications,” Journal of Internet Services and Applications, vol. 5, no. 1, pp. 1–17, 2014.
- [456] S. Spinner, G. Casale, F. Brosig, and S. Kounev, “Evaluating approaches to resource demand estimation,” Performance Evaluation, vol. 92, pp. 51–71, 2015.
- [457] W. Iqbal, “Minimalistic adaptive resource management for multi-tier applications hosted on clouds,” in Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pp. 2546–2549, IEEE, 2012.
- [458] J. Ortigoza, F. López-Pires, and B. Barán, “Workload trace generation for dynamic environments in cloud computing,” arXiv preprint arXiv:1507.00090, 2015.

- [459] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in Grid Computing, 2009 10th IEEE/ACM International Conference on, pp. 50–57, IEEE, 2009.
- [460] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in Network Operations and Management Symposium (NOMS), 2012 IEEE, pp. 1287–1294, IEEE, 2012.
- [461] G. Galante and L. C. E. de Bona, "A survey on cloud computing elasticity," in Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, pp. 263–270, IEEE, 2012.
- [462] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, "QoS guarantees and service differentiation for dynamic cloud applications," Network and Service Management, IEEE Transactions on, vol. 10, no. 1, pp. 43–55, 2013.
- [463] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, "Dynamic load management of virtual machines in cloud architectures," in Cloud Computing, pp. 201–214, Springer, 2009.
- [464] R. F. Van der Wijngaart, T. G. Mattson, and W. Haas, "Light-weight communications on intel's single-chip cloud computer processor," ACM SIGOPS Operating Systems Review, vol. 45, no. 1, pp. 73–83, 2011.
- [465] L. Schubert, "Dynamicity requirements in future cloud-like infrastructures." Invited Speaker, EuroCloud CLASS Conference, Available at http://videlectures.net/classconference2012_schubert_infrastructures/, 2012. [Online: accessed 9-January-2017].
- [466] S. Kumar, T. Cucinotta, and G. Lipari, "A latency simulator for many-core systems," in Proceedings of the 44th Annual Simulation Symposium, ANSS '11, (San Diego, CA, USA), pp. 151–158, Society for Computer Simulation International, 2011.
- [467] J. M. Montanana, M. Koibuchi, H. Matsutani, and H. Amano, "Balanced dimension-order routing for k-ary n-cubes," in Parallel Processing Workshops, 2009. ICPPW '09. International Conference on, pp. 499–506, Sept 2009.
- [468] C. Baaij, J. Kuper, and L. Schubert, "Soosim: Operating system and programming language exploration," in 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time System (WATERS 2012) (G. Lipari and T. Cucinotta, eds.), pp. 63–68, Giuseppe Lipari, 2012.

Appendix A

Implementation of HARD3 Modules in OMNeT++

OMNeT++ stands for Objective Modular Network Testbed in C++. It is a discrete event simulation tool designed to simulate computer networks, multi-processors and other distributed systems associated with GUI simulation library debugging and tracing. Its applications can be extended to modeling other systems as well. It has become a popular network simulation tool in the scientific community as well as in industry over the years. A model network consists of “nodes” connected by “links”. The nodes representing blocks, entities, modules, etc, while the link representing channels, connections, etc. The structure of how fixed elements (i.e. nodes) in a network are interconnected together is called the topology. OMNeT++ uses the NETwork Description (NED) language, thus allowing for a more user friendly and accessible environment for creation and editing. It has a human-readable textual topology and also uses the same format as that of a graphical editor. OMNeT++ allows for the creation of a driver entity to build a network at run-time by program. One of the most important factors in any simulation is the programming language, which is C/C++ based. Since it possesses advantages and it is also freeware for the research community, we have selected the OMNeT++ simulator in our simulation studies

The component model that is used in OMNeT++ operates as following: each cComponent exports a set of input and output cGates. The output gates can be connected to the input gates of other components through cChannels which allow arbitrary cMessage objects to be exchanged between the cComponents they connect. To communicate between two modules, one needs to create a message of the right kind and send it on an output gate assuming that the module that is connected on the other side of the output gate knows how to handle the incoming message. In practice, though, some of the components that make up a simulation (for example, the per-host InterfaceTable component that maintains the list of network interfaces that exist within a node) do not define any input or output gates: the other components that are present in the simulation access them directly through function calls without sending messages. This architecture is very convenient to enable distributed simulations because the simulation runtime has explicit information about the set of objects that can send messages to each other and the delays that each connection applies on these messages.

A.1 Simulation Process in OMNET++

An OMNeT++ model consists of the following parts:

- NED language topology description(s) which are files with the .ned extension.
- Message definitions, in files with .msg extension.
- Module implementations and other C++ source code, in .cc files (or .cpp, on Windows)

In order to build an executable simulation program, first we need to translate the MSG files into C++, using the message compiler (opp-msgc). After this step, the process is the same as building any C/C++ program from source: all C++ sources need to be compiled into object files (.o files (using gcc on Mac, Linux) or mingw on Windows) and all object files need to be linked with the necessary libraries to get an executable or shared library. Figure A.1 gives an overview of the process of building and running simulation programs in OMNET++. In each simulation run (snapshot), the following steps are followed, as shown in Figure A.1.

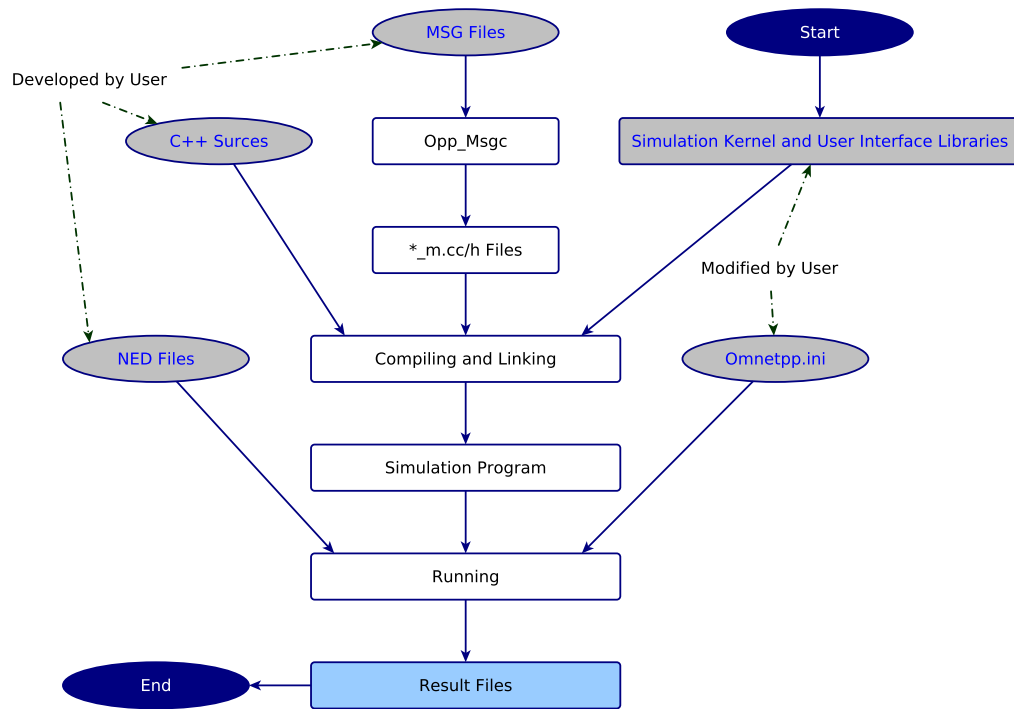


Figure A.1: Overview of the Simulation Process in OMNET++.

A.2 HARD3 Components

Figure A.2 demonstrates the HARD3 simulation components which is developed for the scope of this dissertation work. The following provides a brief description of some of the most important developed modules:

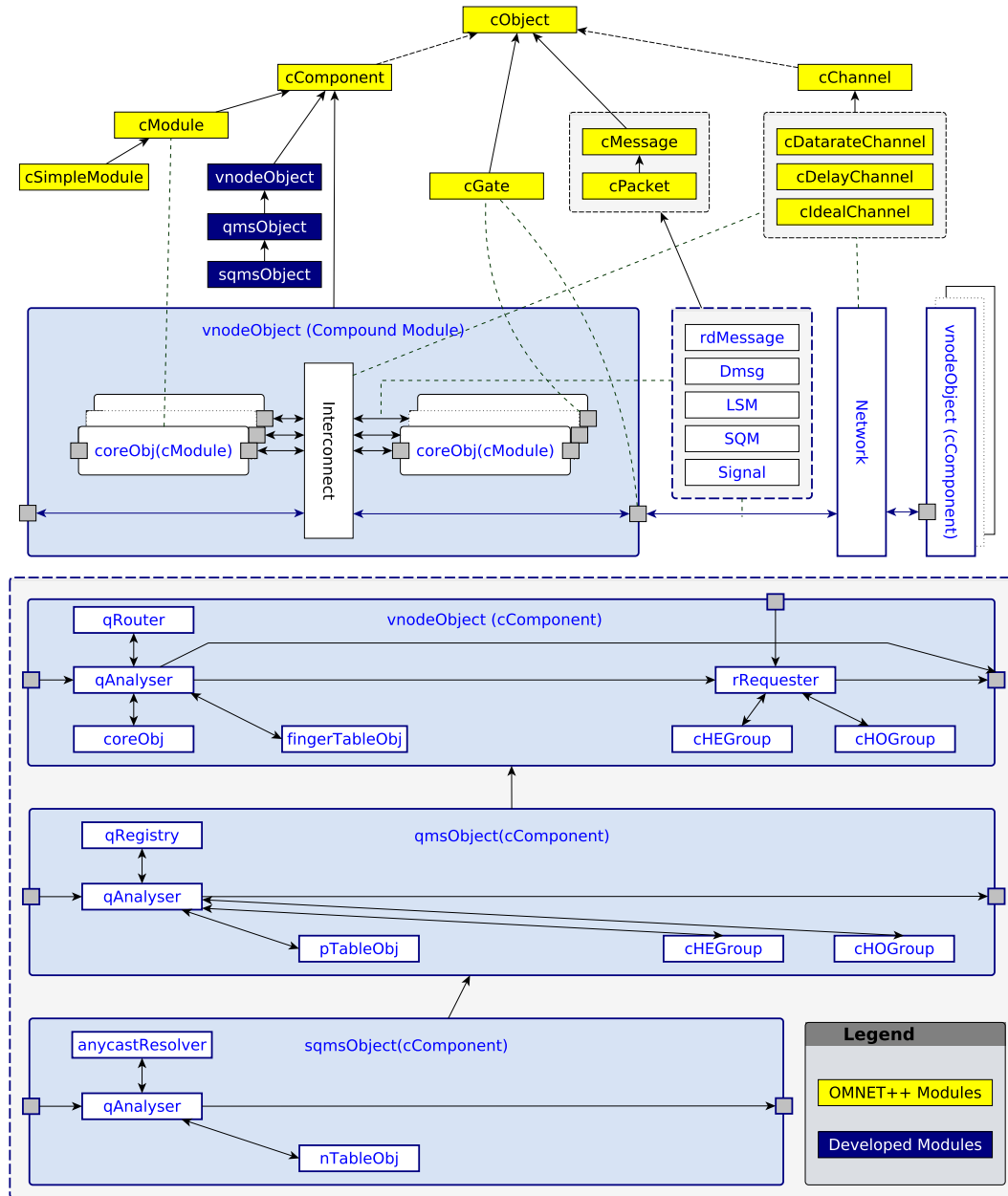


Figure A.2: Class Hierarchy for HARD3 Module Development in OMNET++ (Simulation Component of HARD3).

The **vnodeObject**, **qmsObject** and **sqmsObject** are the compound modules (originally derived from **cComponent**) which include a set of other simple modules as well as the interconnection and network description. These components represent *vnodes* with different module roles while they provide different capabilities and functionalities. The **qmsObject** component is derived from **vnodeObject** and inherits all the properties and functionalities of the **vnodeObject** component. In a similar way, the **sqmsObject** component is derived from the

qmsObject component and inherits all of its features.

The coreObj is derived from cModule and provides a representation of a single processing core containing internal memory hierarchy and the connection to the interconnect. Multiple instantiations of the coreObj are interconnected through a given description of the interconnection network. Thus they provide a representation of a many-core processor. The many-core processors are also connected through the description of the network topology. The aforementioned components leverage several other sub-modules to provide different tasks and functionalities (e.g. qRegistry is responsible for registration and management of a given main-query). Among them qAnalyzer, qRouter, fingerTableObj, rRequester, cHEGroup, cHOGGroup, qRegistry, pTableObj, nTableObj and anycastResolver are some of the most important sub-modules which all derived from cSimpleModule of OMNET++.

Index

- ACO, [47](#)
- AI, [47](#)
- ALU, [68](#), [80](#), [81](#), [91](#), [198](#)
- AN, [96](#), [98–101](#), [103](#), [104](#), [107](#), [119](#), [121](#), [124](#), [126](#), [130](#), [143](#), [167](#)
- ANN, [47](#)
- ASL, [16](#)
- ATNoG, [2](#)
- AVL, [16](#)
- BCA, [47](#)
- BF, [59](#)
- BFS, [41](#), [47](#)
- BOINC, [29](#), [31](#)
- BRW2, [181–183](#), [187](#)
- Cell BE, [60](#)
- CP, [49](#)
- CPU, [1](#), [10](#), [24](#), [28](#), [29](#), [45](#), [60](#), [61](#), [68](#), [74](#), [75](#), [78](#), [94](#), [95](#), [113](#), [114](#), [129](#)
- DAML, [35](#), [36](#)
- DFS, [41](#)
- DHT, [17–24](#), [37](#), [57–59](#), [61](#), [96](#), [97](#), [113](#), [115](#), [119](#), [122](#), [124](#), [126](#), [131](#), [142](#), [143](#), [181](#), [187](#)
- DIS, [24](#)
- DNS, [38](#), [48](#)
- DOS, [80](#), [94](#), [95](#), [99](#), [144](#), [200](#)
- DoS, [25](#)
- DPT, [98](#), [127](#), [134](#), [169](#), [182](#), [185](#)
- ElCore, [144](#), [145](#), [161](#), [199](#), [200](#)
- FALKON, [29](#), [31](#)
- FMCS, [144](#)
- FPGA, [60](#)
- FRW2, [181–183](#), [185](#), [187](#)
- FTR, [169](#), [170](#)
- GFLOPS, [4](#)
- GIIS, [16](#), [61](#)
- GIS, [14](#), [15](#)
- GMD, [15](#), [16](#)
- GPU, [24](#), [60](#), [94](#)
- GRIS, [16](#), [61](#)
- HARD, [4](#), [6](#), [93–96](#), [131](#), [136](#), [160](#), [161](#), [168](#), [171–173](#), [177](#), [180](#), [181](#), [198–200](#)
- HARD2, [181](#), [183–185](#), [187](#)
- HARD3, [95](#), [96](#), [99](#), [119](#), [120](#), [123](#), [124](#), [132](#), [137](#), [143–145](#), [163](#), [169–172](#), [175](#), [178](#), [180–185](#), [187](#), [199](#), [200](#)
- HDL, [67](#)
- HPC, [1–3](#), [9](#), [28](#), [53](#), [55](#), [56](#), [60](#), [65](#), [67](#), [94](#), [144](#), [145](#), [161](#), [199](#)
- HTC, [28](#), [29](#), [31](#), [65](#)
- JCS, [3](#), [4](#), [96](#)
- LDCE, [9](#), [10](#), [22](#), [24](#), [53](#)
- LLH, [85](#), [90](#)
- LN, [96](#), [99](#), [101](#), [103](#), [104](#), [107](#), [108](#), [119](#), [121](#), [124–126](#), [130](#), [167](#)
- MAS, [47](#)
- MBFS, [41](#)
- MDS, [14–16](#), [37](#), [60–62](#)
- MIPS, [4](#)
- MTC, [28](#)
- NED, [165](#)
- NIC, [113](#), [165](#)
- NoC, [60](#), [73](#)
- OIL, [35](#)
- OS, [4](#), [197](#), [200](#)
- P2P, [9](#), [15](#), [17](#), [18](#), [22](#), [24](#), [25](#), [27](#), [32](#), [34](#), [37](#), [38](#), [46](#), [58](#), [60](#), [61](#), [67](#), [94](#), [109](#)
- PA, [16](#)
- PM, [148](#), [149](#)

PRW2, [181–185](#), [187](#)
 QMS, [98](#), [113–115](#), [119–126](#), [128–131](#), [133–135](#),
 [137](#), [138](#), [140](#), [143](#), [166](#), [167](#), [169](#), [171–](#)
 [173](#)
 QoS, [18](#), [32](#), [33](#), [37](#), [52](#), [53](#), [56](#), [57](#), [61](#), [144](#),
 [161](#), [199](#)
 QREG, [130](#), [134](#)
 QROUT, [130](#), [134](#), [138](#)

 RAM, [10](#), [61](#), [82](#), [113](#)
 RCaT, [16](#), [17](#)
 RCT, [126–128](#)
 RD, [2](#), [17](#), [18](#), [165–167](#)
 RDF, [35](#), [36](#)
 RDP, [1](#)
 RM, [145](#)
 RMC, [145](#), [149](#), [152](#)
 RP, [119](#), [120](#), [122](#), [123](#), [126](#), [130](#), [147](#), [161](#)
 RP-QMS, [95](#), [96](#), [99](#), [119](#)
 RP-SQMS, [95](#), [96](#), [98](#)
 RR, [95](#), [96](#), [98](#), [99](#), [119](#), [120](#), [123](#), [131](#), [143](#),
 [144](#), [147](#), [152](#), [161](#)

 S(o)OS, [2](#), [3](#), [95](#), [200](#)
 SIMD, [4](#), [23](#), [80](#)
 SISD, [80](#)
 SLA, [32](#), [33](#), [37](#)
 SMP, [73](#)
 SN, [96–100](#), [119](#), [121](#), [124](#), [139](#), [140](#), [143](#), [167](#),
 [181](#), [185](#)
 SOA, [32](#), [161](#), [199](#)
 SOC, [32](#)
 SON, [23](#), [37](#)
 SoOSim, [200](#)
 SoR, [95](#), [126](#), [132](#), [135](#), [137](#), [138](#), [169](#), [172–175](#),
 [178](#), [182](#)
 SP, [49](#)
 SQMS, [98](#), [114](#), [119](#), [121–123](#), [131](#), [134](#), [138](#),
 [140](#), [143](#), [181](#), [184](#)

 TDD, [187](#)
 TORQUE, [29](#), [31](#)
 TTL, [26](#), [41](#), [51](#), [55](#), [134](#)

 UDDI, [35](#)

 VEE, [33](#)
 VEEH, [33](#)
 VEEM, [33](#)
 VM, [33](#)
 vnode, [95–98](#), [119](#), [122](#), [125](#), [126](#), [130](#), [135](#),
 [143](#), [173](#), [187](#)
 VO, [16](#), [24](#), [128](#)

 WRMS, [31](#)
 WSDL, [35](#)

 XDR, [16](#)
 XML, [16](#), [35](#), [61](#)